PROYECTO FIN DE MÁSTER EN

SISTEMAS INTELIGENTES

CURSO 2013-2014

ALGORITMO DE CALIDAD DE EXPERIENCIA PARA TRANSMISIONES DE VÍDEO EN REDES DEFINIDAS POR SOFTWARE

Jesús Antonio Puente Fernández

Director:

Luis Javier García Villalba

Departamento de Ingeniería del Software e Inteligencia Artificial

Convocatoria: Septiembre

Calificación: Sobresaliente

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid(UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "***Algoritmo de Calidad de Experiencia para Transmisiones de Vídeo en Redes Definidas por Software***", realizado durante el curso académico 2013-2014 bajo la dirección de Luis Javier García Villalba en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

_____

Jesús Antonio Puente Fernández

Abstract

Software Defined Networks (SDN) is a new network paradigm that removes the rigidity present on current architectures and improves flexibility and management in networks. SDN decouples the control plane and the data plane in routing devices and establish an open communication interface between them. In addition, SDN proposes a centralized control of the network. OpenFlow is the first SDN standard that has been widely used in different research projects. Furthermore, continued growth in demand of multimedia content over the network requires that infrastructure provides greater speed, flexibility and Quality of Service (QoS). Especially, video streaming occupies a large amount of total information that circulates on Internet. Moreover, Quality of Experience (QoE) is a new concept in multimedia communications has experimented an important development in last years because it has kept in mind the user expectations to measure the quality respect a given service or application. This research project proposes a routing algorithm for video streaming using OpenFlow and a Floodlight controller to provide QoE for the user.

Keywords

Algorithm, Controller, Floodlight, Mininet, Multimedia, OpenFlow, Quality of Experience, Quality of Service, QoE, QoS, Routing, SDN, Software Defined Networks, Streaming, Video.

Resumen

Las Redes Definidas por Software, abreviadamente SDN (de sus siglas en inglés), es un nuevo paradigma que elimina la rigidez presente en las arquitecturas actuales y mejora la flexibilidad y administración de las redes. SDN desacopla el plano de control y el plano de datos en los dispositivos de encaminamiento y establece una interfaz abierta de comunicación entre ellas. Además, SDN propone el control centralizado de la red. OpenFlow es el primer estándar SDN que ha sido ampliamente utilizado en diferentes proyectos de investigación. Por otra parte, el continuo crecimiento en la demanda de contenido multimedia a través de la red requiere que la infraestructura brinde mayor velocidad, flexibilidad y calidad de servicio. En especial, la transmisión de vídeo ocupa gran cantidad del total de información que circula por Internet. Además, el nuevo concepto de Calidad de Experiencia (abreviadamente, QoE) en comunicaciones multimedia ha tenido un importante desarrollo en los últimos años, ya que toma en cuenta las expectativas del usuario para medir la calidad respecto a un determinado servicio o aplicación. En el presente trabajo de investigación se propone un algoritmo de encaminamiento para transmisiones de vídeo utilizando OpenFlow y controlador Floodlight que mejora la QoE al usuario.

Palabras clave

Algoritmo, Calidad de Experiencia, Calidad de Servicio, Controlador, Encaminamiento, Floodlight, Mininet, Multimedia, OpenFlow, QoE, QoS, Redes Definidas por Software, SDN, Transmisión, Vídeo. .

Agradecimientos

Sírvase estas primeras líneas para agradecer muy especialmente a mi tutor Javier García Villalba, por haberme orientado en la selección de mi proyecto fin de máster y por su valiosa ayuda en el desarrollo del mismo, que como tutor supo indicarme el camino para su buena finalización. También le deseo agradecer por estar siempre disponible, así como proporcionarme sus proyectos de investigación y amplia información bibliográfica que me han servido de base y guía en este proyecto.

Asimismo, quisiera dar a Leonardo Valdivieso y Ana Sandoval las gracias por su ayuda incondicional en todo momento.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Lista de acrónimos

API	*Application Program Interface*
AS	*Autonomous System*
AT&T	*American Telephone and Telegraph*
CLI	*Command Line Interface*
CLR	*Common Language Runtime*
CPU	*Central Processing Unit*
DASH	*Dynamic Adaptive Streaming over HTTP*
DHCP	*Dynamic Host Control Protocol*
DNS	*Domain Name System*
E2E	*End to End*
FTM	*Flow Table Manager*
HD	*High Definition*
HTTP	*HyperText Transfer Protocol*
IaaS	*Infratructure as a Service*
IETF	*Internet Engineering Task Force*
IO	*Input Output*
IoC	*Inversion of Control*
IoT	*Internet of Things*
IP	*Internet Protocol*
IPC	*Inter Process Communication*
LAN	*Local Area Network*
LOBUS	*Load-Balancing over UnStructured networks*

M2M	*Machine to Machine*
MAC	*Media Access Control*
MNS	*Managed Network Services*
MPD	*Media Presentation Description*
MPLS	*MultiProtocol Lable Switching*
NAT	*Network Address Translation*
NCP	*Network Control Point*
NI	*Network Inspector*
OF	*Optimization Functions*
OFELIA	*OpenFlow in Europe: Linking Infrastructure and Applications*
OM	*OpenFlow Module*
PID	*Process IDentifier*
QFF	*QoE Fairness Framework*
QoE	*Quality of Experience*
QoS	*Quality of Service*
SDN	*Software Defined Networking*
TCP	*Transfer Control Protocol*
UF	*Utility Functions*
UI	*User Interface*
VM	*Virtual Machine*
XML	*eXtensible Markup Language*

1. INTRODUCCIÓN

1.1. Evolución de las Arquitecturas de Red Tradicionales

Las tecnologías de comunicación han sufrido una gran evolución desde la década de los 80 hasta llegar al término actual de Redes Definidas por Software (abreviadamente SDN, correspondiente a las siglas de su nomenclatura inglesa *Software-Defined Networks*). Sin embargo, hay dos conceptos que forman la base del desarrollo de esta tecnología. Estos avances son:

- Control Centralizado, Redes Activas y Virtualización de Redes.
- Separación del Plano de Control del Plano de Datos.

A continuación se analizan los aspectos fundamentales de cada uno.

Los orígenes del **control centralizado de la red** se remontan a principios de los 80 y se centran en la forma del control de las primeras redes telefónicas de American Telephone and Telegraph (AT&T), que aún sigue vigente. Inicialmente, el control que sufría este tipo de redes se llevaba a cabo en el mismo plano de datos ya que se transportaban en el mismo canal. Por ejemplo, en la red telefónica, en donde la voz y el control de las señales de ésta se realizaban sobre el mismo canal. Ciertas frecuencias, como por ejemplo 2,6 MHz, enviadas externamente en este canal podían inicializar el teléfono y truncar las líneas. Algunos pulsos en la línea podían ser usados para encaminar llamadas o cambiar opciones de los circuitos. Esto ofrecía muchas ventajas en términos de simplicidad; sin embargo, era bastante frágil, inseguro y vulnerable. En la Figura 1.1 se muestra una Blue Box de Steve Wozniak, a través de la cual se realizaba el envío de señales, pulsos y frecuencias sobre el canal de la red telefónica para controlarla.

Figura 1.1: Blue Box de Steve Wozniak [COU13]

A mediados de la década de los 80, AT&T dio un giro particular separando los planos de control y datos en un componente llamado el *Network Control Point* (NCP). Este modelo fue desarrollado solamente para la red telefónica. En su caso particular, lo que se separó fue la señal de control de la señal de voz. Por tanto, la idea fue que todas las señales de control irían al NCP. Éste se comunicaba con una base de datos que tenía información adicional sobre los clientes. Los beneficios de esta particular tecnología fueron la habilidad de desplegar servicios específicos bajo demanda y, algo más importante, la posibilidad de implementar nuevos servicios rápidamente. Estas ventajas eran prácticamente imposibles con la arquitectura anterior.

Otro avance importante que han experimentado las redes fue la aparición de las **redes activas** en los años 90. Las redes activas [SJSZRP98] [SJSZRP00] permiten realizar tareas personalizadas en los paquetes que viajan a través de los conmutadores. Un ejemplo de redes activas son los *middleboxes* ó "cajas" en la red que realizan tareas de cortafuegos, funciones proxy, servicios de aplicaciones y tareas personalizadas en el tráfico de la red. Esta tecnología fue desarrollada debido a la dificultad para probar nuevos servicios de red en una infraestructura. Las Redes Definidas por Software (SDN) tienen la misma motivación que las redes activas, es decir, acelerar la innovación. En las redes

activas hay dos enfoques diferentes: la encapsulación y los conmutadores programables. La encapsulación consiste en que cada mensaje o cada paquete lleva un programa y los nodos activos desplegados en el camino evalúan dicho código. Así, el código puede ser enviado a un entorno y ejecutarse en un conmutador con soporte para esta tecnología. El otro enfoque son los conmutadores programables, en los cuales los comandos son almacenados en los elementos de red que realizan procesamiento personalizado de paquetes. El procesamiento depende de los valores en el campo de la cabecera de los paquetes entrantes.

Es importante resaltar la aparición y el concepto de la **virtualización de redes**. El término virtualización se ajusta a la representación de una o más topologías lógicas de red en la misma infraestructura física subyacente. Hay diferentes instancias de virtualización de redes, algunas de ellas aparecieron en los 90 como las *Virtual Local Area Networks* (VLAN). Los beneficios que presenta la virtualización son múltiples como, por ejemplo, la compartición de recursos. Usando virtualización se pueden instanciar múltiples encaminadores lógicos en un nodo físico. En otras palabras, se pueden instanciar varias redes virtuales en la misma infraestructura. Sin embargo, esta compartición requiere de la habilidad de aislar los recursos en términos de capacidad de procesamiento de *Central Processing Unit* (CPU), memoria, ancho de banda, tablas de encaminamiento, etc.

En todos los dispositivos encargados de encaminar los paquetes a través de la red se pueden distinguir lógicamente dos planos: el plano de control y el plano de datos.

El plano de control se refiere a la lógica que controla el comportamiento de la red. Ejemplos de este plano son los protocolos de encaminamiento, las configuraciones de *middleboxes* en la red como la de un *firewall*, balanceador de carga, etc. En otras palabras, se puede definir como el cerebro de la red.

Por otro lado, el plano de datos reenvía el tráfico aplicando la lógica del plano de control. Ejemplos del plano de datos son el reenvío de un paquete por un puerto, modificar la cabecera de un paquete IP, la lectura de direcciones MAC, etc. Este plano es normalmente implementado en hardware aunque existen diseños basados en software.

Pero, ¿por qué separar el plano de datos del de control? Una razón de la separación es la de permitir evolucionar y desarrollarse independientemente. Además, en un plano de control separado los dispositivos de red pueden ser controlados por un software de alto nivel independientemente del fabricante del hardware de la red. Adicionalmente, las actualizaciones de los algoritmos, protocolos y políticas pueden ser centralizadas y personalizadas por cada administrador de red.

Un caso de estudio de las ventajas de la separación entre el plano de control y el de datos es un centro de cálculo, donde es relativamente común la necesidad de mover máquinas virtuales de una localización física a otra. Por ejemplo, las instalaciones de Yahoo están compuestas de alrededor de 20.000 servidores en un clúster, resultando en total unas 400.000 máquinas virtuales que necesitan comunicarse entre sí [COU13].

1.2. Redes Definidas por Software

Las Redes Definidas por Software es un nuevo paradigma que reúne los dos avances analizados en el epígrafe anterior. En primer lugar, SDN separa el plano de datos del plano de control en los dispositivos de red. En segundo lugar, SDN propone un control centralizado del plano de control mediante una aplicación de software de alto nivel. De esta manera, los administradores pueden tener un control centralizado y programable del comportamiento del tráfico dentro de la red, sin requerir acceso físico a los dispositivos hardware de red.

Seguidamente se analiza las características básicas de este tipo de redes.

El Protocolo de Internet (*Internet Protocol*, IP) está basado en redes que fueron inicialmente construidas sobre la noción de Sistemas Autónomos Distribuidos (*Autonomous System*, AS) donde para enviar un mensaje desde una fuente A hacia un destino B no es necesario que desde el principio se conozca todo el camino. En la arquitectura actual, el mensaje (paquete IP) va circulando desde un dispositivo hacia otro hasta llegar a su destino. Dicho dispositivo de red tiene un plano de datos y un plano de control integrado y cerrado, que lee la cabecera del mensaje y ejecuta un algoritmo de encaminamiento para determinar el siguiente salto por donde enviar el mensaje, es decir, el camino entre fuente y destino se va estableciendo por medio de los dispositivos de red disponibles.

Por su parte, SDN, al ofrecer separación de planos y un control centralizado, puede establecer el camino más óptimo de la fuente hacia el destino en función de las condiciones de la red. En este paradigma un controlador central recibe la situación actual de la red (número de dispositivos, número de enlaces, ancho de banda disponible, …) y establece el camino entre fuente y destino. Este camino se envía a los encargados de transmitir el mensaje evitando que cada elemento tenga que volver a recalcular la ruta. En otras palabras, el controlador enviará las órdenes a los conmutadores y éstos únicamente transmitirán el paquete salto a salto por el camino previamente asignado. Además, SDN propone que el controlador tenga una interfaz abierta, de tal manera que los usuarios puedan programar sus propias aplicaciones y servicios de red y sean implementados directamente en toda la red.

SDN ofrece importantes ventajas respecto a las tradicionales tecnologías de red. A continuación se analizan los principales avances y campos de aplicación.

Una de las aplicaciones de SDN es la mejora en el rendimiento de los centros de datos. Por ejemplo, el concepto de infraestructura como servicio ó IaaS

(acrónimo del inglés *Infratructure as a Service*). En este caso, las organizaciones e individuos usan recursos de máquinas virtuales (*Virtual Machine*, VM) bajo demanda. A pesar de que físicamente las máquinas virtuales se encuentran en sitios diferentes, la conexión de dichos recursos tiene que ser transparente para el usuario, es decir, la infraestructura tiene que tener la capacidad de soportar la movilidad de las máquinas virtuales dentro de diferentes centros de datos sin afectar al servicio prestado a los clientes. En este contexto, las Redes Definidas por Software pueden ser programadas para coordinar el transporte de información de manera dinámica, sin necesidad de continuamente configurar los dispositivos de red individuales, como sucede actualmente.

Entre otras aplicaciones de SDN se encuentra el concepto Internet de las Cosas (*Internet of Things,* IoT) o Máquina a Máquina (*Machine to Machine*, M2M). Estos términos hacen referencia al incremento de dispositivos que continuamente se conectan a la red y transmiten información. Estos dispositivos incluyen a "cosas" intercambiando información entre sí. Por ejemplo, domótica, coches, puertas, luces, monitores de salud personal, etc. En este escenario, la conectividad de dispositivos que continuamente cambian de posición, requiere que la red brinde alta conectividad y pueda modificar sus rutas dinámicamente. SDN, gracias a la administración dinámica y a su visión global de la red, puede implementar nuevos algoritmos que brinden una mejor eficiencia y conectividad.

1.3. Objetivos de la Investigación

Tendencias como la movilidad del usuario, la virtualización de servidores y los nuevos modelos de negocios, aplicaciones y servicios *online* plantean demandas importantes en seguridad, velocidad y rendimiento que las arquitecturas de red convencionales no pueden satisfacer. En este contexto, SDN propone una nueva arquitectura que permite transformar las redes tradicionales en plataformas dinámicas de prestación de servicios.

Las actuales tendencias muestran que el futuro de las redes se basará cada vez más en software, lo que acelerará el ritmo de la innovación. Las Redes Definidas por Software desacoplan el plano de control del plano de datos en los dispositivos de red. De esta manera se realiza una abstracción de la infraestructura para que pueda ser directamente programable por software. Asimismo, fomenta el uso de herramientas de virtualización de redes, permitiendo al personal de Tecnologías de la Información (*Information Technology*, IT) gestionar de manera óptima sus servidores, aplicaciones y servicios. Las Redes Definidas por Software prometen transformar las redes estáticas actuales en plataformas programables flexibles con la inteligencia necesaria para asignar los recursos de forma dinámica.

Por otro lado, el continuo crecimiento de los dispositivos conectados a la red ha incrementado exponencialmente la cantidad de información que circula por la misma. Servicios multimedia *on-line* (youtube, VoIP, *e-commerce*) requieren que los servicios de telecomunicaciones brinden mayor velocidad, seguridad y flexibilidad. En especial, la transmisión de vídeo ocupa gran cantidad del total de información que circula por Internet. Sin embargo, la mayoría de protocolos no ofrecen distinción entre los diferentes tipos de tráfico.

Además, muchos servicios de Calidad de Servicio (*Quality of Service,* QoS) o el emergente concepto de Calidad de Experiencia (*Quality of Experience,* QoE) son propietarios y requieren que toda la infraestructura pertenezca a un determinado proveedor. La QoE toma en cuenta la percepción del usuario respecto a un determinado servicio o aplicación. En otras palabras, la QoE analiza el grado de satisfacción del cliente

En el presente trabajo se analizan las Redes Definidas por Software, proponiéndose un algoritmo de optimización de la QoE para transmisiones multimedia de vídeo a través de una red utilizando la tecnología SDN.

1.4. Estructura del Trabajo

El resto del trabajo está organizado en 6 capítulos con la estructura que se comenta a continuación:

El Capítulo 2 describe las Redes Definidas por Software (SDN), analizando su evolución en los últimos años y las oportunidades y retos que presenta dicha tecnología.

El Capítulo 3 presenta la arquitectura SDN denominada OpenFlow. OpenFlow es el estándar SDN más utilizado por la comunidad científica que ofrece un protocolo abierto de comunicación entre el controlador y el conmutador. Se muestran los elementos de la arquitectura OpenFlow, haciendo énfasis en el conmutador OpenFlow y en el protocolo OpenFlow como canal de comunicación entre el conmutador y el controlador.

El Capítulo 4 analiza el Sistema Operativo de Red, sus características, ventajas y las principales herramientas disponibles en la actualidad. En otras palabras, se analizan los principales tipos de controladores que actualmente se utilizan en las Redes Definidas por Software. Se clasifican en grupos en cuanto al lenguaje en el que son implementados acompañados de las características propias de cada uno.

El Capítulo 5 presenta el diseño e implementación de un algoritmo mejorar la calidad de vídeo utilizando Redes Definidas por Software o, dicho de otra forma, un algoritmo para el aumento de la QoE en el envío de vídeo a través de una red con conmutadores OpenFlow desde un *host* cliente hasta el servidor, la propuesta original de este trabajo de investigación. Previamente se resumen los trabajos relacionados hasta el momento más importantes utilizando SDN aplicados a diferentes campos como el tratamiento de la carga de datos, QoS y envío de vídeo.

En el Capítulo 6 se detallan los experimentos realizados para evaluar la eficacia de los algoritmos propuestos y los resultados obtenidos.

Finalmente, el Capítulo 7 muestra las principales conclusiones de este trabajo y las líneas futuras de investigación que se pueden derivar del mismo.

2. REDES DEFINIDAS POR SOFTWARE

2.1. Introducción

El nacimiento de nuevos servicios y aplicaciones *on-line*, tanto en terminales fijos como en dispositivos móviles han hecho de las redes de comunicación un punto estratégico, tanto en empresas, instituciones y hogares. La continua evolución de estos servicios y la creciente información que circula en internet han traído retos imprevistos a los desarrolladores y empresas. En especial, los nuevos dispositivos que, gracias a los avances en *Micro-Electro-Mechanical Systems* (MEMS), automáticamente guardan, procesan y envían información con datos relevantes relacionados con las actividades humanas a través de la red. Este tipo de dispositivos, principalmente constituidos por sensores y actuadores (RFID, dispositivos Bluetooth, redes de sensores, sistemas embebidos, …) han dado lugar al nacimiento de nuevos conceptos y paradigmas como es el de IoT.

En 2011 el número de dispositivos conectados en el planeta sobrepasó al número de habitantes. Actualmente, existen 9 billones de dispositivos conectados y se espera una cifra de 24 billones para el 2020 [GBMP13]. Estos dispositivos utilizan diferentes formas de conectarse a la red; entre otras, la infraestructura de red tradicional. Sin embargo, los equipos y protocolos de red tradicionales no fueron diseñados para soportar un alto nivel de escalabilidad, alta cantidad de tráfico y movilidad. Las actuales arquitecturas resultan poco eficientes y presentan limitaciones importantes para satisfacer estos nuevos requerimientos.

La infraestructura encargada de transmitir la información procedente de dispositivos IoT (encaminadores, conmutadores, redes 3G-4G, puntos de acceso) tiene que adaptarse a nuevos servicios post-PC (VoIP, Virtualización, QoS, Computación en la Nube, Aplicaciones de IoT) y, al mismo tiempo,

brindar seguridad, escalabilidad, rapidez y disponibilidad, entre otros. Algunos esfuerzos como SENSE [SSI14], *Internet of Things-Architecture* (IoT-A) [ELIP14] o *Cognitive Management Framework for IoT* [VGSKF13], así como nuevos protocolos como el DDRP [SZMM13] han tratado de obtener una conectividad más inteligente entre los elementos de red. Sin embargo, es posible que no sean la mejor opción para cada uno dominios de aplicación y dispositivos en particular (*Smart Grid, Intelligent Transportation, Smart Home, Health Care, Environmental Monitoring, …*). Por esta razón, en los últimos años ha surgido la idea de personalizar el comportamiento de la red y dar flexibilidad a los usuarios para utilizar los recursos de red según sus necesidades. Más aún, el desarrollo de algoritmos para la toma de decisiones en redes IoT requiere que diferentes métodos (algoritmos genéticos, redes neuronales, algoritmos evolutivos y otras técnicas de inteligencia artificial) puedan ser implementados rápidamente en los equipos de red de forma dinámica sin necesidad de esperar un estándar.

SDN es una arquitectura de red que elimina la rigidez presente en las redes tradicionales. Su estructura permite que el comportamiento de la red sea más flexible y adaptable a las necesidades de cada organización, campus o grupo de usuarios. Además, su diseño centralizado permite recopilar información importante de la red y usarla para mejorar y adaptar sus políticas dinámicamente. El desarrollo de SDN en los últimos años ha impulsado nuevos conceptos, como es el sistema operativo de red (*Network Operating System*, NOS), tratando de emular el avance que se ha tenido en sistemas de computación. Gracias a esta herramienta se ha logrado probar SDN en múltiples proyectos (*Home Networking, Data Centers, Multimedia*, entre otras iniciativas). De igual manera, SDN ha impulsado el diseño de modelos que finalmente integran y logran convergencia entre arquitecturas que tradicionalmente son independientes (WiFi – 4G – LTE). Sin embargo, todas estas oportunidades están aún lejanas de ser implementadas globalmente en equipos de producción. Temas importantes como la convergencia con redes

actuales, escalabilidad, rendimiento, seguridad, etc., son retos importantes que deben superarse para ser posicionados en el mercado.

2.2. Separación del Plano de Datos del Plano de Control

La idea de transmitir información entre dos puntos a través de una red hizo necesario el diseño de protocolos de comunicación (TCP/IP, HTTPS, DNS) y la fabricación de equipos especializados en la transmisión de información. Dichos equipos han evolucionado dando lugar a una gran variedad de dispositivos (*hub, switch, router, firewall, middlebox*, …). Esto ha causado un incremento exponencial en el número de dispositivos conectados.

Todos estos dispositivos encargados de transmitir información tienen características similares en su diseño y fabricación. En primer lugar, existe un hardware especializado en el tratamiento de paquetes (plano de datos) y, sobre el hardware, funciona un sistema operativo (generalmente Linux) que recibe la información del hardware y ejecuta una aplicación de software (plano de control). El software contiene miles de líneas de código y su objetivo es determinar el siguiente salto que debería tomar un paquete para llegar a su destino. El programa sigue las reglas definidas por un protocolo específico (actualmente existen unas 7000 RFCs) o alguna tecnología propia del fabricante. Los equipos modernos también analizan la información de los paquetes en búsqueda de información maliciosa o intrusiones (cortafuegos, sistemas de detección de intrusos). Sin embargo, todo el software o tecnología que se utiliza en la fabricación de estos dispositivos es rígido o simplemente cerrado para el administrador de red. El administrador está limitado a configurar únicamente algunos parámetros, generalmente a través de comandos de bajo nivel usando una interfaz de comandos (CLI). Por otro lado, cada nodo es un sistema autónomo que busca el siguiente salto que debe tomar un paquete para llegar a su destino. Algunos protocolos (OSPF, BGP) permiten que los nodos compartan información de control entre sí, pero únicamente con sus vecinos inmediatos y

de manera muy limitada, con el fin de evitar carga adicional en el tráfico de red. Esto significa que no existe una visión global de la red como un todo. Si el administrador necesita controlar y modificar un camino determinado, el administrador tiene que jugar con parámetros, prioridades o utilizar artilugios para lograr el comportamiento esperado en la red. Cada cambio en la política de red requiere la configuración individual, ya sea directa o de forma remota de cada uno de los equipos. Esta rigidez hace muy complicada la implementación de políticas de red de alto nivel que sean adaptativas, es decir, que sean flexibles y reaccionen dinámicamente según las condiciones de la red.

Al igual que los sistemas operativos evolucionan y se adaptan a las nuevas necesidades y tendencias tecnológicas (soporte multi-CPU, multi-GPU, 3D, soporte pantalla táctil, entre otras), la adaptabilidad de la red a nuevos requerimientos (VLAN, IPv6, QoS, VoIP) se materializa por medio de protocolos o RFCs. Sin embargo, a diferencia del sistema operativo que, gracias a su separación hardware, permite la continua actualización de aplicaciones o directamente su actualización completa, en el área de redes el período de diseño de una nueva idea hasta su publicación en un protocolo y posterior instalación en los equipos puede durar algunos años. Algunos servicios propietarios de los fabricantes requieren que toda la infraestructura de la red sea de la misma firma para funcionar apropiadamente. Esta limitación favorece la dependencia de una tecnología o firma específica.

2.3. Características

El concepto de SDN no es nuevo y completamente revolucionario, sino que más bien surge como el resultado de contribuciones, ideas y avances en la investigación en redes. En [ONF14] se determinan 3 estados importantes en la evolución de SDN: redes activas (de mediados de los 90 a principios de 2000), separación de los planos de datos y de control (2001-2007) y el API OpenFlow y NOS (2007-2010). Todos estos aspectos se analizan a continuación.

La dificultad de los investigadores para probar nuevas ideas en una infraestructura real y el tiempo, el esfuerzo y los recursos necesarios para estandarizar estas ideas en la *Internet Engineering Task Force* (IETF) hizo necesario dar cierta programabilidad a los dispositivos de red. Las redes activas proponen una interfaz programable o *network* API que abre al usuario los recursos individuales de cada nodo como procesamiento, recursos de memoria, procesamiento de paquetes y permitían incluir funcionalidades personalizadas a los paquetes que circulaban a través del nodo. La necesidad de utilizar diferentes modelos de programación en los nodos dio el primer paso para la investigación en virtualización de las redes, así como el desarrollo de *frameworks* o plataformas para el desarrollo de aplicaciones en el nodo. La *Architectural Framework for Active Networks* v1.0 [ONF14] [Ca99] contiene un sistema operativo de nodo (*Node Operating System*, NodeOS) compartido, un grupo de ambientes de ejecución (*Execution Environments*, EEs) y aplicaciones activas (*Active Aplications*, AAs). The NodeOS administra los recursos compartidos, mientras que los EE definen a una máquina virtual para las operaciones de paquetes. Las AA operan dentro de un EE y brindan el servicio extremo a extremo. La separación de paquetes a cada EE depende de un patrón en la cabecera de los paquetes entrantes al nodo. Este modelo fue utilizado en la plataforma *PlanetLab* [Pl14], en donde los investigadores realizaban experimentos en ambientes virtuales de ejecución y los paquetes eran demultiplexados a cada ambiente virtual en función su cabecera. Estos avances resultaron importantes, especialmente en la investigación de arquitecturas, plataformas y modelos de programación en redes. Sin embargo, su aplicabilidad en la industria fue limitada y criticada principalmente por sus limitaciones en rendimiento y seguridad. El trabajo presentado en [WoTu01] es un esfuerzo para brindar un mayor rendimiento a las redes activas; el *Secure Active Network Environment Architecture* [AAKS98] intentó mejorar su seguridad.

El crecimiento exponencial de los volúmenes de tráfico que circulan por la

red acarreó la necesidad de mejorar la gestión y de utilizar mejores funciones de administración como es el manejo de los caminos o enlaces que circulan en la red (ingeniería de tráfico), predicción de tráfico, reacción y recuperación rápida a problemas en la red, entre otros. Sin embargo, el desarrollo de estas tecnologías se han visto fuertemente limitadas por la estrecha unión entre el hardware y software de los equipos de red. Además, el continuo incremento en las velocidades de enlace (*backbones*) hizo que todo el mecanismo de transmisión de paquetes (*packet forwarding*) fuese concentrado en el hardware, separando el control o la administración de red a una aplicación de software. Dichas aplicaciones funcionarían mejor en un servidor, ya que presenta mayores recursos de procesamiento y memoria que los disponibles en un solo dispositivo de red. En este sentido, el proyecto ForCES (*Forwarding and Control Element Separation*) [YDAG04] estandarizado por la IETF (RFC 3746) estableció una interfaz entre los planos de datos y de control en los nodos de red. El software *SoftRouter* [LNRS04] utilizaba esta interfaz para instalar *forwarding tables* en el plano de datos de los *routers*. Asimismo, el proyecto *Routing Control Plataform* (RCP) [CCFRS05] propuso un control lógico centralizado de la red. De esta manera se facilitaba la administración y se daba capacidad de innovación y programación de red. RCP tuvo una aplicabilidad inmediata, ya que aprovechó un protocolo de control existente, BGP (*Border Gateway Protocol*), para instalar entradas en las tablas de encaminamiento de los *routers*.

Con la separación de los planos de datos y control se desarrollaron arquitecturas "*clean-slate*" como es el proyecto 4D [GHM05] o Ethane [CFPL07]. La arquitectura 4D propone una arquitectura de 4 capas según su funcionalidad: *data plane, discovery plane, dissemination plane* y *decision plane*. Por su parte, el proyecto Ethane [CFPL07] propone un sistema de control centralizado de enlaces para redes empresariales. Sin embargo, la necesidad de conmutadores personalizados basados en Linux, OpenWrt, NetFPGA con soporte para el protocolo Ethane, hizo difícil su aplicabilidad. Actualmente, el

protocolo OpenFlow [MABP08] es el más utilizado en la comunidad científica y ha sido la base para la realización de diferentes proyectos. Empresas como Cisco también han presentado una propuesta de nueva arquitectura denominado *Cisco Open Network Environment* (Cisco ONE).

Simplificando el análisis previo, el término SDN propone algunos cambios a las redes de hoy en día. En primer lugar, establece la separación o desacople de los planos de datos y de control, permitiendo su independiente evolución y desarrollo. En segundo lugar, propone que el plano de control sea lógicamente centralizado teniendo de esta manera una visión global de la red. Finalmente, se instauran interfaces abiertas entre los planos de control y de datos. Las diferencias entre estas arquitecturas se presentan en la Figura 2.1.

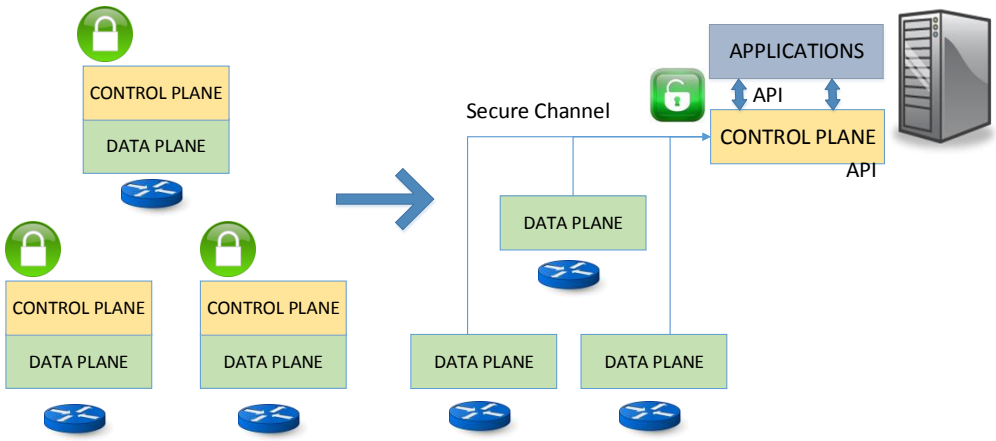


Figura 2.1: Comparación entre la arquitectura tradicional y SDN.

La programabilidad que ofrece SDN a la red puede compararse como las aplicaciones móviles que hoy en día son ejecutadas sobre un sistema operativo (Android, Windows Mobile). Dichas aplicaciones utilizan los recursos del móvil (GPS, acelerómetro, memoria) gracias al API que ofrece el sistema operativo. De la misma forma, el administrador de red gracias a las API disponibles (propietarias o abiertas) en el controlador, puede gestionar y programar los recursos de la red según las necesidades de los usuarios.

2.4. Aplicaciones SDN

SDN brinda la capacidad de modificar el comportamiento de la red según las necesidades del usuario. Es decir, SDN por sí misma no resuelve ningún problema en concreto, sino que brinda una herramienta más flexible para gestionar de mejor manera las redes. Con el fin de probar las ventajas de esta arquitectura, la comunidad investigadora ha presentado múltiples proyectos de interés. A continuación se resumen algunas de estas aplicaciones.

- ***Home networking***. En el incipiente campo de la IoT, la gestión de los dispositivos y los recursos de red en redes residenciales resulta todo un desafío debido al número de usuarios y dispositivos conectados a un mismo punto (usualmente, un punto de acceso). En [KSXFE11] [KF13] se presenta una implementación de un sistema basado en Openflow que permite la monitorización y administración del acceso de usuarios a Internet basados en *usage caps*, es decir, una capacidad limitada de datos por usuario o dispositivo. El sistema permite visibilidad sobre los recursos de red, administración de acceso a nivel de usuario, grupo de usuarios, dispositivo, aplicación u hora del día e, incluso, el intercambio de capacidad de acceso con otro usuario. El control y monitorización de la red se realiza a través de una interfaz amigable de usuario *Kermit* y la administración de la capacidad y políticas de red por medio del *lenguage Resonance* [NRFC09].

- **Seguridad**. La seguridad también puede ser mejorada debido a la visión global de la red. La seguridad no puede basarse únicamente en la seguridad de los *hosts* (antivirus), ya que cuando éstos se encuentran comprometidos dichas defensas no son efectivas. En [RMTF09] se presenta el sistema *Pedigree* como alternativa de seguridad en el tráfico que circula por la red corporativa. Este sistema, basado en Openflow, permite al controlador analizar y autorizar el tráfico y conexiones que circula en la

red. Los *hosts* tienen un módulo de seguridad a nivel de kernel *(tagger)* que no se encuentra bajo el control del usuario. Este módulo etiqueta las conexiones que solicitan enviar información a través de la red (procesos, archivos, etc.). Dicha etiqueta se envía hacia el controlador (*arbiter*) al inicio de la comunicación. El controlador analiza y acepta o rechaza la conexión según sus políticas. Una vez que se autoriza la conexión, las tablas de flujo correspondientes se instalan en el conmutador. *Pedigree* presenta mayor resistencia a una variedad de ataques de evasión como los gusanos polimórficos. El sistema agrega una mayor carga al tráfico de red y al *host*. Sin embargo, esta carga no es mayor al de un software antivirus común.

- **Redes móviles**. Los dispositivos de la infraestructura de redes portadoras móviles (*mobile carrier networks*) comparten similares limitaciones que las redes de computadores. Las redes portadoras de igual forma siguen estándares y protocolos, por ejemplo los propuestos por el *Third Generation Partnership Project* (3GPP), así como implementaciones propietarias específicas de los vendedores. En este punto el paradigma SDN y su modelo basado en flujos (*flow-based forwarding model*) puede aplicarse a este tipo de infraestructura ofreciendo mejores herramientas. *Software-Defined Mobile Network* (SDMN) [PWH13] es una arquitectura que permite a los operadores apertura, innovación y programabilidad sin depender de un fabricante exclusivo o proveedores de servicios *Over The Top* (OTT). Este modelo consta de 2 elementos: *MobileFlow Forwarding Engine* (MFFE) y el *MobileFlow Controller* (MFC). MFFE es el plano de datos simple, estable y de alto desempeño. Tiene una estructura más compleja que un conmutador OpenFlow ya que soporta funcionalidades adicionales de portadoras como son la tunelización de capa 3 (por ejemplo GTP-U y GRE), funcionalidades de nodos de redes de acceso y de carga flexible. El MFC corresponde al plano de control de alta capacidad, en donde se desarrollan las aplicaciones de redes móviles. De igual manera, se establecen interfaces

3GPP para interconectarse con diferentes tipos de *Mobile Management Entity* (MME), *Serving Gateway* (SGW) *o Packet Data Network Gateway* (PGW).

- **Multimedia**. Los múltiples servicios *on-line* multimedia como, por ejemplo, transmisión de contenido en tiempo real, requieren altos niveles de eficiencia y disponibilidad por parte de la infraestructura de red. Según estudios presentados por CISCO [TZE13], para el 2017 el 73% de todo el tráfico IP (público y privado) será tráfico de vídeo IP (en 2012 era del 60%). Además, en los últimos años ha tomado fuerza el término de QoE [PaPe12], que intenta redefinir la QoS tomando en consideración el nivel de aceptación del usuario a un determinado servicio o aplicación multimedia. En este sentido, SDN permite optimizar las tareas de administración multimedia. Por ejemplo, en [KSDM12] se mejora la experiencia QoE a través de la optimización de rutas. Esta arquitectura consiste de los elementos: el QoS *Matching and Optimization Function* (QMOF) que lee los parámetros multimedia y determina la configuración apropiada para el enlace y el *Path Assignment Function* (PAF) que mantiene actualizada la topología de la red. En el caso de una degradación de la calidad en los enlaces, el sistema automáticamente modifica los parámetros de los enlaces tomando en cuenta las prioridades de los usuarios. Asimismo, el proyecto *Openflow-assisted QoE Fairness Framework* QFF [GEBMR13] busca las transmisiones multimedia que se encuentran en la red y ajusta dinámicamente las características de la transmisión en función de los dispositivos terminales y los requerimientos de la red.

- **Confiabilidad y Recuperación**. Uno de los problemas comunes en las redes tradicionales es la dificultad para recuperarse cuando falla un enlace. El tiempo de convergencia se ve afectado por la limitada información que posee el nodo para recalcular una ruta. En algunos casos, se requiere inevitablemente la intervención del administrador para que manualmente

restablezca los enlaces en la red. En este punto SDN, gracias a su visión global, permite la personalización de los algoritmos de recuperación. En [SSCP12] se propone un sistema basado en Openflow que utiliza los mecanismos de restauración y protección para buscar un camino alternativo. En restauración el controlador busca otro camino cuando recibe la señal de caída de enlace. Por su lado, el método de protección se anticipa a un fallo y calcula previamente un camino alternativo. Por otro lado, al igual que el mal funcionamiento de un conmutador o encaminador puede afectar gravemente la disponibilidad de la red, en SDN el mal funcionamiento del controlador (fallo del NOS, ataque DDoS, error de la aplicación) puede ocasionar un colapso de toda la red. En este sentido, la confiabilidad de la red puede garantizarse por medio de controladores de respaldo (*backup*). Sin embargo, es necesario que tanto el controlador principal como el secundario tengan actualizada y coordinada la misma información de control y configuración. El componente *CPRecovery* [FBMP12] es un mecanismo de *backup* primario que permite la replicación de información entre el controlador principal y de respaldo. El sistema usa la fase de replicación para mantener actualizado el controlador *backup* y la fase de recuperación que inicia el controlador de respaldo al momento de detectar un error en el controlador principal.

- **Virtualización**. El concepto de virtualización en redes tiene similitud con virtualización en sistemas de cómputo, donde diferentes sistemas operativos pueden compartir recursos hardware, es decir, en virtualización de redes se intenta que múltiples redes virtuales puedan operar sobre una misma infraestructura, cada una con una topología y lógica de encaminamiento propia. Inicialmente, las tecnologías VLAN y redes privadas virtuales permiten que varios usuarios compartan recursos de la red. Sin embargo, la separación se controla sólo por el administrador de red con parámetros limitados (puerto del conmutador) y únicamente opera

con protocolos de red conocidos. Con la separación de los planos de control y de datos que soporta SDN, las posibilidades de crear redes virtuales más avanzadas es prometedora. Por ejemplo, Flowvisor [GYAC09] es una plataforma de virtualización que utiliza OpenFlow [MABP08] y se ubica lógicamente entre las capas de control y encaminamiento. Flowvisor [GYAC09] actúa como un proxy transparente entre los controladores y conmutadores. Luego crea un plano virtual y transparente según las políticas establecidas por el administrador, asegurando aislamiento en términos de ancho de banda, *flowspace* y carga en el CPU del conmutador. El usuario puede observar y controlar únicamente su propio *slice*. Adicionalmente, es posible volver a dividir un slice virtual y tener de esta manera una jerarquía de redes virtualizadas, tal y como se muestra en la Figura 2.2.

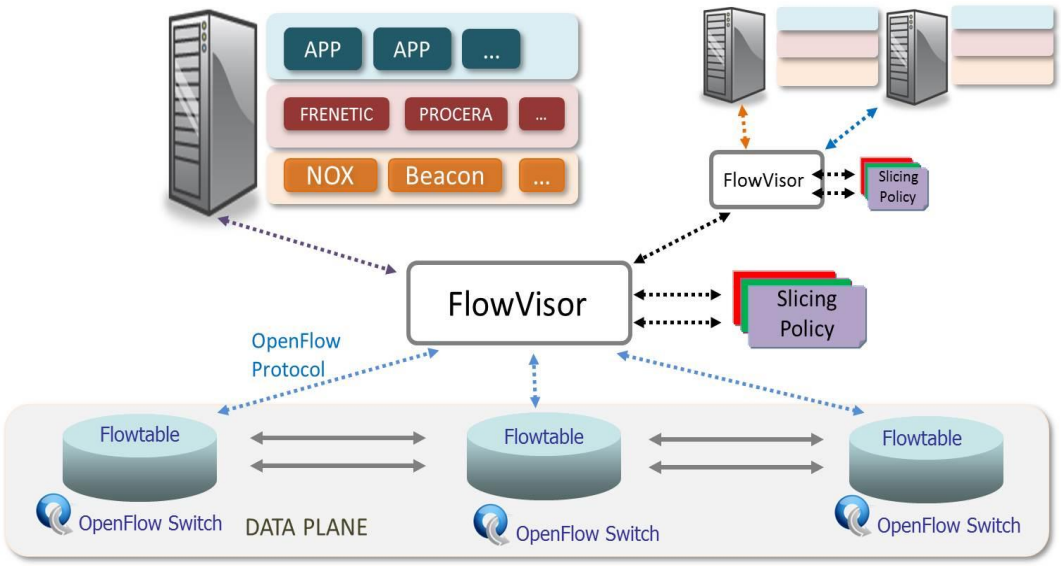


Figura 2.2: Protocolo OpenFlow, Virtualización y Sistemas Operativos de Red.

En [GYAC10] hay una demostración de cuatro exitosos experimentos usando Flowvisor [GYAC09] (balanceador de carga, transmisión de *video streaming*, ingeniería de tráfico y experimentos de hardware experimental), cada uno con su propio *slice*. Sin embargo, los experimentos muestran algunos problemas por resolver: interacción inesperada con otros

dispositivos de red instalados, incremento del tráfico de *broadcast* emitidos por dispositivos no OpenFlow y algunas violaciones del aislamiento en CPU, especialmente cuando un *slice* añade una regla de encaminamiento que es enviado por el conmutador a través de un camino lento.

Otro aspecto importante es la integración entre las diferentes operaciones de red y virtualización de sistemas operativos S.O. En virtualización de S.O., las diferentes máquinas virtuales VMs requieren una capa de acceso de red que permita interconexión entre VMs y fuera de él y además soporte funciones de red comunes a una capa física tradicional. El modelo común es establecer comunicación entre nodos virtuales y el NIC físico implementando un típico encaminamiento de capa L2 (*switching*) o L3 (*routing*). Esto dificulta la administración de la red en ambientes virtuales, por ejemplo al momento de migrar VMs entre diferentes servidores físicos. En este enfoque, SDN y virtualización de redes puede ayudar a lograr estos objetivos.

Open *vSwitch* [PPAC09] es un conmutador basado en software diseñado para ambientes virtuales. Este conmutador exporta una interfaz para un minucioso control de la red. Adicionalmente, tiene una partición lógica para el plano de encaminamiento basado en un flexible motor de encaminamiento basado en tablas. El plano de encaminamiento tiene una interfaz externa y puede ser administrado por ejemplo a través de OpenFlow [MABP08]. Con esta abstracción, el controlador puede obtener una vista lógica de múltiples Open vSwitches ejecutándose en servidores separados físicamente.

Otra aplicación interesante en virtualización es la *Virtual Network Migration* (VNM). En redes tradicionales, la migración o el cambio en un nodo de la red requiere la re-configuración y re-sincronización de los protocolos de encaminamiento. Esto causa altos retardos y pérdida de paquetes. En este

ámbito, el uso de nodos virtuales puede reducir significativamente el tiempo de inactividad.

En el sistema VNM propuesto en [PFCMC10], el controlador SDN crea nuevas entradas *flow* para el nuevo conmutador y redirecciona el camino del nodo inicial hacia el siguiente. Luego, el controlador elimina las entradas *flow* del conmutador antiguo permitiendo ser retirado con seguridad. Los resultados de experimentos muestran un tiempo total de migración de 5 ms sin aparente pérdida de paquetes. Más aún, el sistema podría ser reconfigurado dinámicamente para ubicar redes virtuales en diferentes nodos físicos según la hora del día o la demanda de tráfico para ahorrar energía (*green networks*).

2.5. Retos de la tecnología SDN

Las ventajas que ofrece SDN como tecnología aplicable a las redes de producción masiva se encuentran cercanas pero no disponibles. Más aún, existen algunos retos en términos de seguridad, escalabilidad, confiabilidad, entre otros aspectos, que deben superarse con el fin de ser consideradas aceptables para usuarios comerciales. A continuación se analizan brevemente estos aspectos.

Como se explicó anteriormente, la separación de los planos de datos y de control permite su independiente desarrollo y evolución. En el plano de datos la velocidad de procesamiento de paquetes depende principalmente de la tecnología utilizada en el hardware, ya sea *Application-Specific Integrated Circuits* (ASIC), *Application-specific Standard Products* (ASSP), *Field Programmable Gate Array* (FPGA) o *multicore* CPU/GPP. Por su parte, en el plano de control el rendimiento depende principalmente del hardware y del NOS (Beacon, POX, Floodlight). Sin embargo, el bajo desempeño de uno de los dos puede ocasionar problemas significativos, como son la pérdida o retraso de paquetes,

comportamientos erróneos de la red o denegación de servicio. Por esta razón, es necesario que los diseños de hardware y software para componentes de redes SDN tengan balance en rendimiento, coste y facilidad de desarrollo.

Por otro lado, Openflow utiliza los recursos de hardware comunes en los equipos actuales mediante el uso de tablas de flujo. Sin embargo, SDN puede extenderse más allá de las tablas de flujo y utilizar otros recursos adicionales que ofrece actualmente el hardware [VBG13]. La integración y estudio de nuevas funcionalidades entre el plano de control y el plano de datos personalizado es un campo recién abierto. Aplicaciones como cifrado, análisis, clasificación de tráfico y dispositivos como *middleboxes*, procesadores de paquetes personalizados, entre otros, pueden integrarse y ser usados eficientemente con la tecnología SDN. Por otro lado, el número y la ubicación de los controladores dentro de la red es una pregunta abierta. El análisis presentado en [HSM12] expone que los elementos determinantes para la elección del número y ubicación del controlador son la topología de la red y el rendimiento que se espera de la red.

La seguridad es otro aspecto fundamental que también debe ser tomado en cuenta. Por ejemplo, no todos las aplicaciones de red deben contar con los mismos privilegios de acceso [SSCF13]. Es necesaria la asignación de perfiles, autenticación y autorizaciones para acceder a los recursos de la red. Por otro lado, Openflow establece el uso opcional de TLS (*Trasport Layer Security*) como herramienta de autenticación entre el controlador y el conmutador. Sin embargo, no existen especificaciones claras que brinden seguridad para sistemas de múltiples controladores que intercambian información entre sí y con los conmutadores. Asimismo, debido a que Openflow establece que un paquete desconocido sea enviado completamente o su cabecera al controlador, fácilmente se pueden ejecutar ataques de denegación de servicio mediante el envío de múltiples paquetes desconocidos al conmutador.

La transición de arquitecturas actuales hacia arquitecturas SDN es de igual forma un campo abierto. A pesar de que actualmente ya existen equipos con soporte para Openflow (NEC, IBM) en el mercado, es imposible remplazar toda la infraestructura ya instalada. El período de transición requiere de mecanismos, protocolos e interfaces que permitan coexistencia eficiente de ambas arquitecturas. Actualmente existen esfuerzos para lograr este objetivo: la *Open Networking Foundation* ONF publicó el protocolo IF-Config [OMC13] como primer paso para la configuración de equipos con soporte Openflow. De igual manera, el *European Telecommunications Standards Institute* (ETSI) y el *IETF´s Forwarding and Control Element Separation Working Group* (ForCES) trabajan en la estandarización de interfaces para el correcto desarrollo de esta tecnología.

3. ARQUITECTURA OPENFLOW

3.1. Introducción

OpenFlow es un estándar creado por la Universidad de Stanford, inicialmente diseñado para permitir a los investigadores ejecutar protocolos experimentales en las redes de un campus, que provee un mecanismo estandarizado para ejecutar experimentos sin requerir la exposición de la estructura interna de los dispositivos de red. Actualmente, OpenFlow tiene soporte en conmutadores Ethernet comerciales, *routers* y puntos de accesos inalámbricos.

3.2. Arquitectura OpenFlow

La arquitectura OpenFlow propone la existencia de un controlador, un conmutador OpenFlow y un protocolo seguro de comunicación entre ellos. Dichos elementos se muestran en la Figura 3.1.

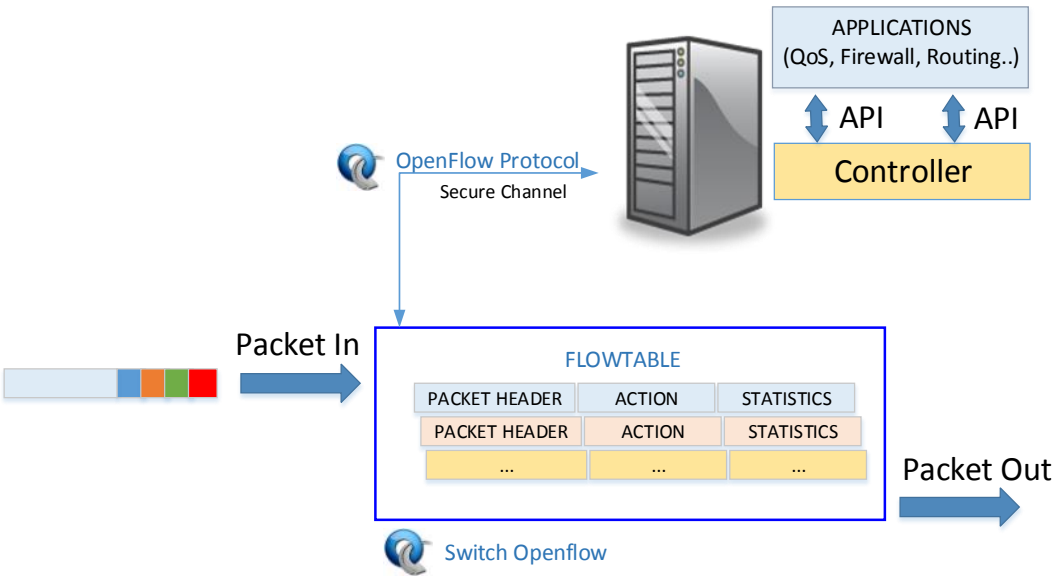


Figura 3.1: Elementos de la arquitectura OpenFlow.

Cada conmutador OpenFlow está formado por tablas de flujo que son administradas desde el controlador. Cada tabla de flujo consta de tres elementos: *packet header*, *action* y *statistics*. El *packet header* es una máscara encargada de seleccionar los paquetes que van a ser procesados por el conmutador. Los campos que se utilizan para la comparación pueden ser indistintamente de la capa 2, 3 o 4 de la arquitectura TCP/IP. En otras palabras, no existe una separación entre capas como sucede en las arquitecturas actuales. Todos los paquetes que llegan al conmutador son filtrados por medio de este método. El número de campos que el conmutador puede procesar depende de la versión del protocolo OpenFlow utilizado. En la versión OpenFlow v1.0 [OSS09], que es la versión más utilizada, existen 12 campos, mientras que la última versión disponible OpenFlow v1.3 define la existencia de 40 campos incluyendo soporte para IPv6.

Una vez que la cabecera de un paquete entrante coincide con el *packet header* del *flowtable*, las acciones correspondientes para esa máscara son ejecutadas por el conmutador. Existen acciones principales y opcionales. Las acciones principales son: reenviar el paquete a un puerto determinado, encapsular el paquete y enviarlo hacia el controlador y descartar el paquete. Finalmente, el campo de *statistics* contabiliza entre otros la información del número de paquetes por cada flujo y se utiliza para fines de administración. En el caso de que la cabecera de un paquete entrante no coincide con el *packet header* del *flowtable*, el conmutador (según su configuración) envía dicho paquete hacia el controlador para su análisis y tratamiento.

3.3. Conmutador OpenFlow

Un conmutador OpenFlow consiste en una tabla de flujo (*flow table*) y un canal externo (*secure channel*) que se conecta al controlador. Estos componentes se pueden apreciar en la Figura 3.2.

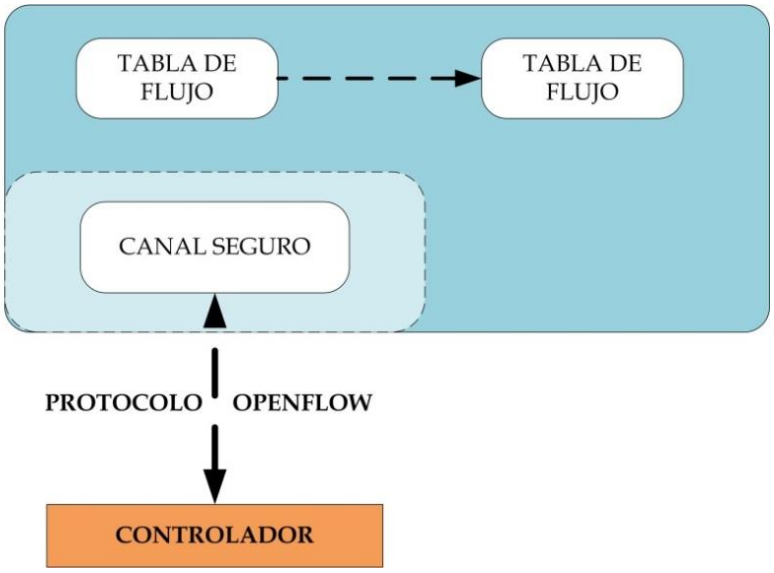


Figura 3.2: Componentes de un conmutador OpenFlow.

El controlador maneja el comportamiento del conmutador a través del canal seguro utilizando el protocolo OpenFlow. El controlador puede añadir, actualizar y borrar información de la tabla de flujo, tanto reactivamente (en respuesta a paquetes) como proactivamente (generando acciones).

Cada tabla de flujo en el conmutador contiene un conjunto de entradas (*flow entries*). Éstas, a su vez, consisten en valores de cabecera (*header values*), contadores de actividad y un conjunto de cero o más acciones para aplicar a los paquetes. Cada vez que entra un paquete, el conmutador compara la cabecera del paquete con las entradas de la tabla de flujo. Si los campos coinciden, las instrucciones asociadas a ese flujo se ejecutan y, en caso contrario, se envía el paquete al controlador por medio del canal seguro. Por tanto, el controlador es responsable de determinar cómo se manejan los paquetes sin entrada de flujo válida. Dichas instrucciones se envían al conmutador para reconfigurar la tabla de flujo, permitiendo que se envíen directamente los siguientes paquetes. Las acciones asociadas a las entradas son: envío del paquete por un puerto determinado, re-escritura de la cabecera del paquete y descartar paquete.

3.4. Tablas OpenFlow

En este apartado se describe los componentes de las tablas de flujo, además del mecanismo de comprobación de coincidencia y manejo de las acciones.

3.4.1. Tabla de Flujo

Una tabla de flujo, tal y como se muestra en la Tabla 3.1, es una estructura que contiene 3 campos:

- *Campos de Cabecera*: se usan para hacer la comprobación de coincidencia de los paquetes entrantes.
- *Contadores*: se utiliza para registrar el número de paquetes coincidentes.
- *Instrucciones*: determinan las acciones que se ejecutarán con los paquetes cuyas cabeceras son idénticas a los campos coincidentes.

Campos de Cabecera	Contadores	Instrucciones

Tabla 3.1: Cabeceras de una tabla de flujo.

La Tabla 3.2 muestra los campos de cabecera que pueden ser utilizados para la comparación con los paquetes entrantes. Cada entrada contiene un valor específico o el valor ANY para un valor arbitrario. Los campos coincidentes pueden ser indistintamente de la capa 2, 3 o 4 de la arquitectura TCP/IP.

Campo	Bits	Aplicable a	Notas
Ingress Port	(depende de la implementación)	Todos los paquetes	Representación numérica del puerto de entrante, empezando en 1.
Ethernet Source Address	48	Todos los paquetes en puertos habilitados	
Ethernet Destination Address	48	Todos los paquetes en puertos habilitados	
Ethernet Type	16	Todos los paquetes en puertos habilitados	Un conmutador OpenFlow es requerido para la comprobar la coincidencia del tipo tanto en el estándar Ethernet como 802.2 con una cabecera SNAP y OUI de valor 0x000000. El valor especial 0x05FF es usado para coincidir con todos los paquetes 802.3 sin cabeceras SNAP.
VLAN Id	12	Todos los paquetes del tipo Ethernet 0x8100	
VLAN Priority	3	Todos los paquetes del tipo Ethernet 0x8100	Campo PCP de VLAN.
IP Source Address	32	Todos los paquetes IP y ARP	Puede ser enmascarado por subred.
IP Destination address	32	Todos los paquetes IP y ARP	Puede ser enmascarado por subred.
IP Protocol	8	Todos los paquetes IP e IP sobre Ethernet. Paquetes ARP	Sólo los 8 bits menos significativos son usados para el *ARP opcode*.
IP ToS Bits	6	Todos los paquetes IP	Especifica un valor de 8 bits y está localizado en los 6 bits superiores de ToS.
Transport Source Port / ICMP Type	16	Todos los paquetes TCP, UDP e ICMP	Sólo los 8 bits menos significativos son usados para tipo ICMP.
Transport Destination Port / ICMP Code	16	Todos los paquetes TCP, UDP e ICMP	Sólo los 8 bits menos significativos son usados para tipo ICMP.

Tabla 3.2: Parámetros utilizados en el Experimento 1 de identificación de la fuente.

El tratamiento que un paquete recibe cuando entra en el conmutador se describe en la Figura 3.3.

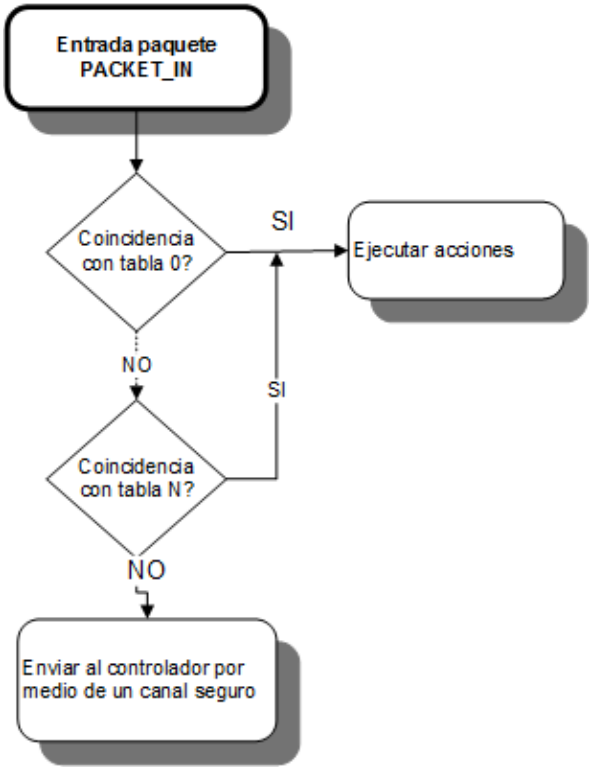


Figura 3.3: Procesamiento de un paquete en un conmutador OpenFlow.

Como se explicó anteriormente, el conmutador compara el paquete con los campos de la tabla de flujo; en caso de coincidencia ejecuta las acciones, actualiza contadores y busca en la siguiente tabla. En caso de que el paquete sea desconocido, el conmutador (según su configuración) descarta o encapsula el paquete y lo envía al controlador.

Las diferentes acciones que se pueden ejecutar se dividen en requeridas y opcionales:

- **Acción Requerida**: *Forward*. Esta acción envía un paquete por un puerto específico (tanto físico como virtual). Los puertos estándar son definidos como: puertos físicos, virtuales (definidos por el controlador) y el puerto LOCAL. Además, los conmutadores OpenFlow soportan estas opciones adicionales de envío:

- ALL: envía el paquete de salida a todos los puertos estándares, menos el puerto de entrada.
- CONTROLLER: encapsula y envía el paquete al controlador.
- LOCAL: envía el paquete a la pila de red del conmutador local.
- TABLE: realiza acciones en la tabla de flujo. Sólo para mensajes *Packet-Out*.
- IN PORT: envía el paquete al puerto de entrada.

- **Acción Opcional:** *Forward*. El conmutador tiene la opción de soportar las siguientes acciones en los puertos virtuales:
  - NORMAL: procesa el paquete usando algoritmos tradicionales (no-OpenFlow) del conmutador. El conmutador comprueba el campo VLAN para determinar si se puede o no enviar el paquete a través de la ruta normal de procesamiento. En caso contrario, el conmutador envía un mensaje indicando que no soporta esta acción.
  - FLOOD: inunda el paquete sobre toda la red, excluyendo la interfaz de entrada.
- **Acción Opcional**: *Enqueue*. Este tipo de acción envía un paquete a través de la cola adjunta a un puerto. El comportamiento de envío está determinado por la configuración de la cola y suele proveer soporte básico QoS.
- **Acción Requerida**: *Drop*. El conmutador descarta todos los paquetes que coinciden con una tabla de flujo configurada sin acciones.
- **Acción Opcional:** *Modify-Field*. Este tipo de acción modifica los valores de las respectivas cabeceras en un paquete.

3.5. Canal OpenFlow

El canal OpenFlow es la interfaz que conecta el conmutador OpenFlow con el controlador. Dicha interfaz es conocida como protocolo OpenFlow y por medio de ella se realizan las siguientes acciones:

- Configura y actualiza el conmutador.
- Recibe eventos procedentes del conmutador.
- Envía paquetes al conmutador.

3.5.1. Protocolo OpenFlow

El protocolo OpenFlow define los siguientes tipos de mensajes entre el conmutador y el controlador: *controller to switch*, *symmetric* y *asynchronous*. Los mensajes tipo *controller to switch* gestionan el estado del conmutador, los *asynchronous* actualizan el control de los eventos de la red y cambios al estado del conmutador. Los *symmetric* son enviados ya sea por el controlador o por el conmutador para iniciar la conexión o intercambio de mensajes. De igual manera, se definen 2 tipos de conmutadores: *Openflow-only* y *Openflow-enabled*, según tengan la capacidad de trabajar únicamente con OpenFlow o puedan también procesar el paquete utilizando algoritmos tradicionales de conmutación o de encaminamiento.

En resumen, el protocolo OpenFlow [OSS09] soporta tres tipos de mensajes. Éstos son:

- **Controlador a conmutador**: iniciado por el controlador y usado para inspeccionar el estado del conmutador.
- **Asíncronos**: iniciado en el conmutador y usado para actualizar la información del controlador cuando se producen eventos en la red y cambios de estado en el conmutador.
- **Simétricos:** iniciados por ambos y enviados sin petición.

3.5.1.1. Mensajes Controlador a Conmutador

Este tipo de mensajes son iniciados por el controlador y pueden o no tener respuesta por parte del conmutador. Estos tipos de mensajes son:

- ***Features***: el controlador solicita las capacidades de un conmutador enviando una petición de características (*features request*). El conmutador responde con una respuesta a la petición (*features reply*).
- ***Configuration***: el controlador está capacitado para cambiar y hacer peticiones sobre parámetros de configuración en el conmutador. El conmutador sólo responde a una petición generada desde el controlador.
- ***Modify-State***: estos mensajes son enviados por el controlador para administrar estados en los conmutadores. Su propósito principal es añadir, eliminar o modificar flujos en las tablas OpenFlow. Además, permiten cambiar las propiedades de los puertos del conmutador.
- ***Read-State***: son usados por el controlador para coleccionar estadísticas del conmutador.
- ***Send-Packet***: estos mensajes son usados por el controlador para enviar paquetes por un puerto específico del conmutador y para reenviar paquetes previamente recibidos.
- ***Barrier***: los mensajes de petición/respuesta de barrera son usados por el controlador para asegurar dependencias que haya tenido o para recibir notificaciones de operaciones completadas.

3.5.1.2. Mensajes Asíncronos

Estos mensajes son enviados sin la petición del controlador al conmutador. Estos mensajes denotan la llegada de un paquete, cambios de estado en el conmutador o errores. Los cuatro principales tipos de mensajes asíncronos son:

- ***Packet-in***: Este mensaje encapsula un paquete y lo envía al controlador

para su procesamiento. Este mensaje es enviado cuando no existe una tabla asociada a la cabecera del paquete.

- ***Flow-removed***: Informa al controlador que un elemento de la tabla de flujo ha sido eliminado del conmutador.
- ***Port-status***: el conmutador envía mensajes de este tipo al controlador para informar sobre cambios de estados en la configuración del puerto.
- ***Error***: el conmutador notifica al controlador de problemas existentes.

3.5.1.3. Mensajes Simétricos

Estos mensajes son enviados bidireccionalmente sin petición. Los mensajes simétricos son:

- ***Hello***: son intercambiados entre el conmutador y el controlador a la hora de establecer la conexión.
- ***Echo***: los mensajes *Echo* pueden ser enviados tanto por el controlador como por el conmutador y se envían siempre en respuesta a una petición. Son usados para medir la latencia o el ancho de banda de una conexión controlador- conmutador.
- ***Vendor***: este tipo de mensajes proveen a los conmutadores OpenFlow una forma estándar de ofrecer funcionalidad adicional dentro del espacio del mensaje OpenFlow.

3.6. Ventajas de OpenFlow

Como se ha comentado anteriormente, OpenFlow fue inicialmente propuesto como alternativa para el desarrollo de protocolos experimentales en campus universitarios, donde se puedan probar nuevos algoritmos sin necesidad de interrumpir o interferir con el funcionamiento normal del tráfico de otros usuarios. Hoy en día, la Open *Networking Foundation* ONF [ONF14] es la

organización encargada de la publicación del protocolo Openflow, entre otros protocolos SDN como, por ejemplo, *OF-Config* [OMC13].

La ventaja que OpenFlow presenta con respecto a protocolos SDN previos radica en que OpenFlow aprovecha elementos y funciones de hardware ya disponibles en la mayoría de los equipos de red. Estos elementos son las tablas de encaminamiento y las funciones comunes como leer la cabecera, enviar el paquete a un puerto, descartar paquete, entre otros. Openflow abre estos elementos y funciones para que puedan ser controlados externamente. Esto implica que basta con una actualización de *firmware* para que el mismo hardware pueda ser ya compatible con Openflow. De esta manera las empresas no necesitan realizar un cambio completo de su hardware para implementar SDN en sus productos y servicios.

El controlador recibe la información de los diferentes conmutadores y configura remotamente las tablas de flujo de los conmutadores. Es en el controlador, donde el usuario puede literalmente programar el comportamiento de la red. A diferencia de las redes activas que proponían un *Node Operating System*, OpenFlow abre la noción de un *Network Operating System* (NOS). En este aspecto, en [FRZ13] se define al NOS como el software que abstrae la instalación del estado en los conmutadores de red de la lógica y aplicaciones que controlan el comportamiento de la red. En los últimos años los NOS han ido evolucionando según las necesidades y aplicaciones de los investigadores y administradores de red.

4. SISTEMA OPERATIVO DE RED

4.1. Introducción

Un Sistema Operativo de Red o NOS (acrónimo del inglés *Network Operating System*) describe a todas las herramientas de software que permiten crear aplicaciones y controlar el comportamiento de la red. Los NOS se clasifican según el lenguaje de programación que utilizan. Cada lenguaje tiene sus ventajas en función de administración de la memoria, soporte multiplataforma y rendimiento. En la Tabla 4.1 se resumen los principales NOS.

Lenguaje	Nombre del Controlador
C/C++	NOX, Trema, MUL
Haskell	Nettle, McNettle, NetCore
Java	Maestro, Floodlight, Beacon
OCaml	Mirage, Frenetic
Python	POX, Pyretic, RYU

Tabla 4.1: NOS en función del lenguaje de programación [E13].

4.2. Evolución de los Sistemas Operativos de Red

El concepto de Sistema Operativo de Red (*Network Operating Systems*, NOS) se basa en la función de un sistema operativo en el campo de computación, es decir, el sistema operativo permite al usuario crear aplicaciones usando abstracción de alto nivel de los recursos de información y de hardware. En SDN algunos autores [RFRS12] [SSCF13] [KF13] han clasificado las abstracciones de los recursos de red como interfaces *southbound* y *northbound* (Figura 4.1).

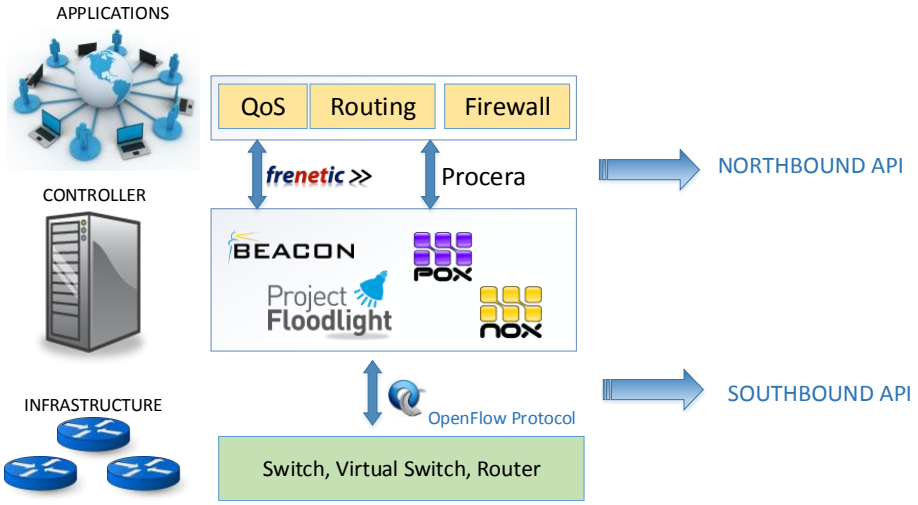


Figura 4.1: NOS e interfaces northbound y southbound.

Las interfaces tipo *southbound* tienen la función de abstraer la funcionalidad de los conmutadores programables y conectarse con el software controlador. Un claro ejemplo de interfaz *southbound* es OpenFlow. Sobre las interfaces *southbound* se ejecuta un NOS. Ejemplos de NOS son: NOX [GKPC08], Beacon [E13], Floodlight [PF14], entre otros. Por otro lado, las interfaces *northbound* permiten crear aplicaciones o políticas de red de alto nivel y transmiten dichas tareas al NOS. Ejemplos de estas interfaces so:n Frenetic [FHFM11] [FGR13], Procera [VKF12] [KF13], Netcore [MFHW12]; McNettle [VW12]. A continuación se analizan los principales NOS e interfaces *northbound*.

El software NOX [GKPC08] es el primer NOS para OpenFlow y está constituido por 2 elementos: procesos del controlador y una visión global de la red. En función del estado actual de la red, el usuario puede tomar decisiones y configurar el comportamiento de la red por medio de dichos procesos. En NOX el tráfico se maneja a nivel de flujos, es decir, todos los paquetes con la misma cabecera, son tratados de manera similar. El controlador inserta, elimina entradas y lee los contadores que se encuentran en las tablas de flujo (*flow tables*) de los conmutadores. Por otro lado, debido a la naturaleza dinámica del tráfico NOX usa eventos (*event handlers*) que son registrados con diferentes prioridades para ejecutarse cuando se produce un evento específico en la red. Los eventos

más utilizados son: *switch join, switch leave, packet received and switch statistics received*. Asimismo, NOX incluye *system libraries* con implementaciones y servicios de red comunes. Finalmente, NOX es implementado en C++ ofreciendo alto rendimiento. Existe una implementación enteramente en Python denominado POX, que proporciona un lenguaje de desarrollo más amigable.

Beacon [E13] es un controlador OpenFlow basado en Java. Su interfaz es simple y sin restricciones, es decir, el usuario puede usar libremente los constructores disponibles en Java (*threads, timers, sockets*, …). Por otro lado, Beacon es un NOS basado en eventos, es decir, el usuario configura los sucesos que monitoriza el controlador. La interacción con los mensajes Openflow del conmutador se realiza mediante la librería *OpenflowJ*, una implementación del protocolo OpenFlow 1.0 [OSS09], y la interfaz *IBeaconProvider* que contiene los *listeners IOFSwitchListener*, *IOFInitializerListener* y *IOFMessageListener*. Beacon tiene soporte *multithreading* y facilita implementaciones APIs importantes (*Device Manager, Topology, Routing, Web UI*), así como la capacidad de iniciar, agregar y terminar aplicaciones sin terminar completamente un proceso (*runtime modularity*).

A pesar que un NOS puede manejar las tablas de flujo de los conmutadores, existen algunos problemas que pueden ocasionar el mal funcionamiento de la red [RFRS12] [SSCF13] [GRF13]. Por ejemplo, el controlador recibe el primer paquete que llega al conmutador y que no tiene un *flow table* asignado. Luego, el controlador lo analiza, asigna acciones y reenvía esas instrucciones al conmutador para que los demás paquetes similares tengan el mismo camino. Sin embargo, durante ese tiempo puede llegar el segundo, tercer o cuarto paquete similar al controlador y ocasionar un funcionamiento errático. En otras palabras, virtualmente existen dos procedimientos en ejecución: uno en el controlador y otro en el conmutador, y dichos procedimientos no se encuentran completamente sincronizados.

Otra limitación es la composición, es decir, si el usuario desea configurar dos servicios diferentes en el mismo conmutador (por ejemplo, encaminamiento y monitorización), se tiene que combinar manualmente ambas acciones en el conmutador, asignar prioridades, mantener la semántica según cada elemento de la red. Esto hace muy difícil el diseño, coordinación y reutilización de librerías. Además, el conmutador tiene que manejar 2 tipos de mensajes simultáneamente: paquetes y mensajes de control. Cualquier descoordinación puede ocasionar que un paquete sea procesado con una política inválida y, consecuentemente, causar un problema de seguridad importante en la red. Por ejemplo, si en una tabla de flujo existen dos entradas con la misma prioridad, el comportamiento del conmutador podría ser no determinístico, ya que la ejecución dependería del diseño del hardware del conmutador. Para superar este tipo de inconvenientes, la comunidad investigadora ha trabajado en el desarrollo de interfaces más simples que interactúen y coordinen el correcto funcionamiento en el conmutador (*northbound*).

Procera [VKF12] [KF13] es un *framework* que permite expresar políticas o configuraciones de red de alto nivel. Esta arquitectura establece diferentes dominios de control y acciones con las que el usuario programa el comportamiento de la red. Los principales dominios de control son: *Time, Data Usage, Status* y *Flow*. Con estos dominios el usuario puede determinar un comportamiento dependiendo, por ejemplo, de la hora del día, cantidad de datos transmitidos, privilegios o grupos de usuarios, tipo de tráfico transmitido, etc. Las acciones pueden ser temporales o reactivas y están expresadas en un lenguaje de alto nivel basado en *Functional Reactive Programming* (FRP) y *Haskell*. En [KF13] se encuentran los detalles de este lenguaje, así como ejemplos del uso en aplicaciones de monitorización y control de usuarios en un campus.

Frenetic [FHFM11] [FGR13] es un lenguaje de alto nivel para redes SDN desarrollado en Python. Está estructurado en 2 sub-lenguajes: *Network Query Language* y *Reactive Network Policy Management Library*. *Network Query Language*

permite al usuario leer el estado de la red. Esta tarea se realiza mediante la instalación de reglas de bajo nivel (*low-levels rules*) en el conmutador que no afectan al funcionamiento normal de la red. Por otro lado, el *Network Policy Management Library* es diseñado en base a un lenguaje para robots, *Yampa* [CNP03] y librerías para programación web en *Flapjax* [MGBC09]. Las acciones usan un constructor tipo *Rule* que contiene un patrón o filtros y lista de acciones como argumentos. Las acciones principales son: enviar a un puerto determinado, enviar paquete al controlador, modificar la cabecera del paquete, y acción en blanco, que se interpreta como descartar el paquete. La instalación de estas políticas se realiza mediante la generación de *policy events* (similar a *querys*), *primitive events* (*Seconds, SwitchJoin SwitchExit, PortChange*) y *listener (Print, Register*). El resultado de experimentos [FHFM11] muestra que *Frenetic* proporciona simplicidad, así como un ahorro significativo en código y menor consumo de los recursos de la red en comparación a NOX.

Una de las ventajas adicionales de este lenguaje es la composición, es decir, se pueden escribir módulos funcionales independientes y el *runtime system* coordina su correcto funcionamiento en el controlador y en el conmutador. Existen 2 tipos de composición: secuencial y paralela. En la composición secuencial la salida de un módulo es la entrada del siguiente. Por ejemplo, un balanceador de carga que primeramente modifica el destino IP de un paquete y luego busca el puerto de salida en función de la nueva cabecera IP. En la composición paralela ambos módulos son ejecutados virtualmente al mismo tiempo en el controlador. Por ejemplo, si el balanceador envía un paquete con destino IP A hacia el puerto 1 y el paquete con destino IP B hacia el puerto 2. Esta composición resultaría en una función que envía los paquetes entrantes por los puertos 1 y 2.

McNettle [VW12] es un controlador diseñado especialmente para ofrecer alta escalabilidad a la red SDN. Esto se logra mediante un conjunto de manejadores de mensaje (*message handlers*), uno por cada conmutador, que tienen una

función que gestiona las variables *switch-local* y *network state* y administra las acciones de suministro de los flujos de la red. El principio es que los mensajes de un conmutador individual se manejen secuencialmente, mientras que los mensajes de conmutadores diferentes sean manejados concurrentemente. De igual manera, *McNettle* intenta que cada mensaje sea procesado en un único *core* CPU. De esta manera, se reduce al máximo el número de conexiones y sincronizaciones *inter-cores*, mejorándose el rendimiento. Las pruebas realizadas en [VW12] muestran que *McNettle* tienen un desempeño *multicore* considerable en comparación a NOX o Beacon.

El controlador propuesto en [GRF13] se basa en la verificación de las políticas establecidas, en lugar de buscar *bugs* monitorizando el funcionamiento del controlador. Para realizar la verificación, en primer lugar se utiliza el lenguaje de alto nivel denominado *Netcore* [MFHW12], en donde se expresa únicamente el comportamiento de la red, más no su modo de implementación en el controlador. Luego, el *NetCore Compiler* expresa dichas políticas en configuraciones a nivel de conmutador (tablas de flujo). La información de las tablas de flujo es analizada nuevamente por el *Verified Run-time System*, que traduce dicha configuración en un nivel de abstracción más bajo denominado *Featherweght Openflow*. *Featherweght Openflow* es un modelo que permite asegurar que las reglas instaladas en el conmutador son consistentes con la tabla de flujo y que, gracias a primitivas de sincronización, aseguran que el funcionamiento del conmutador sea el correcto. Asimismo, en [RFRS12] se presenta la herramienta *Kinetic*, que ofrece mantener consistencia de actualizaciones en la red mediante dos mecanismos: de paquete y de flujo. En el primero de ellos se garantiza que cuando existe una actualización el paquete que circula en por la red se procesa por una misma configuración. En el segundo se asegura que todos los paquetes pertenecientes al mismo flujo (por ejemplo, la misma conexión TCP) sean tratados de forma similar por los conmutadores de la red.

4.3. NOX/POX

NOX es uno de los primeros controladores OpenFlow de código abierto. En este apartado se agrupan NOX y POX debido a que tienen estructura similar, pero un entorno diferente de implementación (NOX es desarrollado en C++ y POX en Python).

NOX incluye una vista sobre la red en la que incluye la topología a nivel de conmutadores, la localización de los usuarios, *hosts*, *middleboxes*, servicios (por ejemplo, *HyperText Transfer Protocol* (HTTP)) y otros elementos de la red. La vista también incluye todos los enlaces entre nombres y direcciones.

La interfaz de programación de NOX es simple, centrándose sobre eventos, espacio de nombres y vista sobre la red.

- **Eventos:** las redes en general no son estáticas, es decir, los flujos llegan y se van, igual que los usuarios y los enlaces. Para alcanzar este cambio de eventos, las aplicaciones de NOX usan un conjunto de *handlers* o manejadores de eventos que están registrados para ejecutarse cuando una interrupción particular sucede. Estos *handlers* son ejecutados en el orden de prioridades (especificado durante el registro del *handler*). El *handler* devuelve un valor indicado a NOX si hay que parar la ejecución de ese evento, continuarlo o pasar dicho evento al siguiente *handler* registrado. Algunos eventos son generados directamente por los mensajes OpenFlow como *switch join, switch leave, packet received* y *switch statistics received*. Otros eventos son generados por las aplicaciones NOX como resultado de un procesamiento de eventos de bajo nivel. Por ejemplo, NOX incluye aplicaciones que autenticarán a un usuario a través de la redirección de tráfico HTTP con el evento *packet received*.

- **Vista de la red y espacio de nombres:** NOX incluye un número de aplicaciones *base* las cuales construyen la vista de la red y mantienen un

espacio de nombres de alto nivel que puede ser usado por otras aplicaciones. Estas aplicaciones manejan la autenticación del usuario y del *host* para obtener los nombres de los *hosts* a través de una monitorización DNS. La vista de la red debe ser consistente y hacerla disponible para todas las instancias de controladores NOX, por lo que la escritura dirigida a ellos se torna compleja. Debido a esto, las aplicaciones NOX sólo deben avisar cuando se detecta un cambio en la red y no para todos los paquetes recibidos.

- **Servicios de alto nivel:** NOX incluye un sistema de librerías para proveer implementaciones eficientes de funciones comunes a muchas aplicaciones de la red. Éstas incluyen un módulo de encaminamiento, clasificación rápida de paquetes, servicios estándar como *Dynamic Host Control Protocol* (DHCP) y DNS, y un módulo de filtrado basado en políticas de la red.

4.3. Floodlight

Floodlight [PF14] es un controlador OpenFlow desarrollado en Java. Aparece como evolución del controlador Beacon [E13], por lo que igualmente comparten similar estructura. Java brinda soporte multiplataforma y sencillez de programación. El usuario puede hacer uso de herramientas típicas de Java, como son *threads*, temporizadores, *sockets*, etc. Floodlight incluye la librería *OpenFlowJ* para trabajar con mensajes OpenFlow. Esta librería es una implementación Java orientada a objetos de la especificación de la versión 1.0 de OpenFlow. Los *listeners* se utilizan para notificar cuando los conmutadores son añadidos o eliminados (*IOFSwitchListener*), y para recibir distintos tipos de mensajes específicos OpenFlow (*IOFMessageListener*). Además, *Floodlight* contiene aplicaciones básicas de referencia las cuales constituyen el *core*. Esta API adicional es la que se muestra en la Figura 4.2.

Figura 4.2: *Pipeline* del *thread IOFMessageListener* de Beacon [E13].

Las aplicaciones básicas disponibles son:

- ***Device Manager*:** muestra los dispositivos que aparecen en la red, incluyendo sus direcciones (Ethernet e IP), fecha de último uso, el conmutador y el puerto que han sido vistos por última vez. El *Device Manager* provee una interfaz (*IDeviceManager*) para buscar dispositivos conocidos y la habilidad de registrarlos para recibir eventos cuando nuevos dispositivos sean incluidos, actualizados o eliminados.
- ***Topology*:** descubre los enlaces entre los conmutadores OpenFlow. Su interfaz (*ITopology*) permite la recuperación de una lista de los enlaces y el registro de eventos para ser notificados cuando los enlaces son incluidos o eliminados.
- ***Routing*:** provee enrutamiento de la capa L2 con el camino más corto entre dos dispositivos de la red. Esta aplicación exporta la interfaz *IRoutingEngine*, permitiendo implementaciones con diferentes lógicas de encaminamiento. La implementación incluida usa todos los métodos de computación de camino más corto entre dos pares. Esta aplicación depende tanto de la *Topology* como del *Device Manager*.
- **Web:** provee una *Web User Interface* (UI) para Floodlight. La aplicación web provee la interfaz *IWebManageable*, permitiendo a los desarrolladores añadir sus propios elementos UI.

Por otro lado, Floodlight contiene funciones adicionales como se muestra en la Figura 4.3. Se presenta un Java API para el desarrollo de aplicaciones que

residen dentro del controlador y requieren alta eficiencia de procesamiento (por ejemplo, procesamiento de paquetes tipo PACKET_IN). Adicionalmente, el REST API (por las siglas de *Representational State Transfer*) está disponible para configuración remota (puerto 8080 por defecto) de los diferentes servicios del controlador. De esta manera, los usuarios pueden crear aplicaciones que invoquen servicios del controlador mediante peticiones web (HTTP REST).

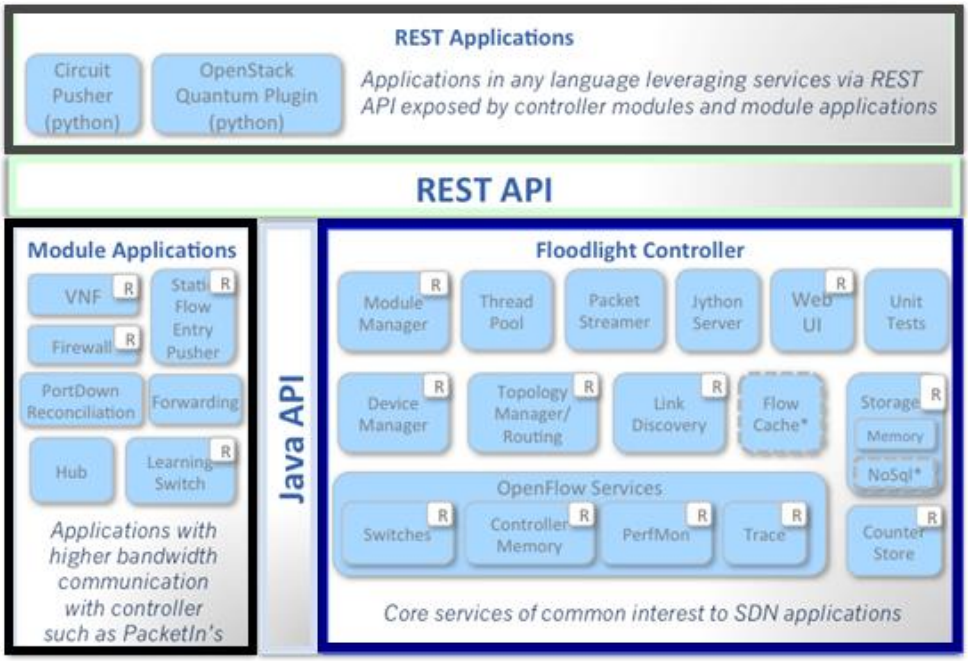


Figura 4.3: API de Floodlight [PF14].

4.3.1. Modularidad en Tiempo de Ejecución

Muchos de los controladores OpenFlow tienen la habilidad de seleccionar qué aplicaciones compilar (modularidad en tiempo de compilación) y qué aplicaciones ejecutar cuando el controlador comienza a actuar (modularidad en tiempo de inicio). Floodlight tiene la capacidad adicional de comenzar y finalizar aplicaciones mientras se está ejecutando, además de añadirlas y eliminarlas (modularidad en tiempo de ejecución) sin que el proceso de Floodlight se termine.

4.4. Pyretic

Pyretic [MRFRW13] es uno de los miembros de la familia de lenguajes de programación SDN. Surge como resultado de uno de los lenguajes de programación de redes como es Frenetic [FO14] pero con sintaxis de Python, ofreciendo una manera simple para expresar políticas de alto nivel, las cuales compilan las reglas de coincidencia OpenFlow. Permite a los programadores especificar políticas en lo que son *paquetes localizados*. Una política está basada en la combinación de un paquete y su localización en la red.

4.4.1. Características

Pyretic ofrece muchas características entre las cuales se encuentran la habilidad de escribir políticas de red como funciones. En otras palabras, Pyretic permite al programador escribir una función que recoge como entrada un paquete, devolviéndolo en diferentes localizaciones en la red.

Pyretic provee predicados booleanos en contraste a otros controladores. Además, a diferencia de manejar reglas de acciones por medio de coincidencias, Pyretic permite la creación de políticas usando conjunciones y predicados como *and* y *not*. En la Tabla 4.2 se muestra una lista de algunas políticas del lenguaje Pyretic.

Sintaxis	Descripción
None	Devuelve el conjunto vacío.
Identidad	Devuelve el paquete original.
Match	Devuelve la identidad si el campo del paquete coincide con un valor particular; en otro caso, devuelve el conjunto vacío.
Mod	Devuelve el mismo paquete pero cambia el campo de la cabecera virtual del paquete a un valor específico o envía el tráfico a través de un puerto de salida particular modificando solamente el atributo que se encarga de dicho puerto en el paquete.
Flood	Devuelve un paquete por cada puerto en el árbol de recubrimiento de la red.

Tabla 4.2: Políticas atómicas del lenguaje Pyretic [MRFRW13].

Por otro lado, Pyretic ofrece registros o campos virtuales en la cabecera del paquete, que habilitan al programador a referirse tanto a las cabeceras actuales como a las virtuales.

Por último, Pyretic provee operadores de composición tanto paralelos como secuenciales para hacer posible que el operador de red escriba políticas complejas compuestas por otras más sencillas.

4.4.2. Predicados

En OpenFlow el conmutador comprueba si la cabecera de paquetes coincide con una regla; en caso contrario, busca la tabla siguiente. Esto hace muy difícil expresar las políticas que envuelven reglas complejas como son conjunciones o negaciones. En este contexto, Pyretic permite analizar los paquetes directamente por medio de predicados. Por ejemplo, en la Figura 4.4 se muestra una conjunción de dos predicados de coincidencia que devolverá los paquetes que tengan como dirección IP destino 10.0.0.3 ó 10.0.0.4.

$$match(dstip = 10.0.0.3) \mid match(dstip = 10.0.0.4)$$

Figura 4.4: Conjunción de predicados

Las cabeceras virtuales del paquete son una manera unificada de representar metadatos en un paquete. En Pyretic un paquete es un diccionario que mapea un nombre de un campo a un valor. Se puede aplicar el predicado de coincidencia a paquetes que llegan a un puerto de entrada particular de un conmutador o que tiene una dirección MAC destino.

Como se ha especificado anteriormente, una composición secuencial realiza la primera operación y, si se cumple, realiza la segunda. Un ejemplo que envía paquetes a un puerto de salida dependiendo de la dirección IP destino es el que se aprecia en la Figura 4.5.

$$match(dstip = 2.2.2.8) \gg fwd(1)$$

Figura 4.5: Composición secuencial de dos predicados.

Por otro lado, la composición paralela realiza ambas operaciones simultáneamente. En la Figura 4.6 se muestra un ejemplo que envía paquetes de un puerto de salida a otro, dependiendo del valor de la dirección IP destino.

$$match(dstip = 2.2.2.8) \gg fwd(1) + match(dstip = 2.2.2.9) \gg fwd(2)$$

Figura 4.6: Composición paralela de dos predicados.

5. OPTIMIZACIÓN DE QOE PARA TRANSMISIÓN DE VÍDEO

5.1. Introducción

Durante los últimos años, gracias al incremento computacional y capacidades de visualización de los dispositivos terminales, se ha incrementado exponencialmente la cantidad de información multimedia que circula por la red. En 2011, según datos de Cisco [TZE13], el tráfico de vídeo constituía el 51% de todo el tráfico en Internet. Siguiendo con esta tendencia, se espera que en 2016 aumente hasta el 55%. El contenido en *High Definition* (HD) también se ha convertido en un gran factor de nivel de calidad: en 2011, por primera vez el tráfico de vídeo en HD sobrepasó la definición estándar.

A pesar de los incrementos en ancho de banda, la infraestructura de red tiene dificultades importantes para evitar congestión en las transmisiones multimedia. Recursos insuficientes de la red puede causar congestión, provocando degradación de la calidad de video (congelado de imágenes, *playback* entre audio-vídeo).

5.2. Trabajos Relacionados

Esta sección analiza los principales trabajos de investigación directamente relacionados con la temática de la investigación desarrollada. Al ser SDN una tecnología incipiente apenas existe literatura al respecto o bien ésta es de difusión restringida, por lo que se ha creído conveniente incluir temas que se llevan a cabo con SDN relacionados con transmisiones de vídeo, Calidad de Servicio (QoS) y Calidad de Experiencia (QoE).

Así, en los últimos años ha tomado atención el concepto de QoE. Gross et al. [GKKW04] definen una escala para medir de este grado con el nombre de *Mean Opinion Score* (MOS), que asigna una escala de valores a la calidad de un

servicio. Un MOS igual a 5 implica una percepción excelente de calidad, mientras un MOS de valor la unidad muestra una pobre calidad. Normalmente, estos valores deberían obtenerse mediante múltiples encuestas a los usuarios durante la duración del servicio, lo que resulta inviable. En este aspecto, es claro que QoE tiene una relación directa con los típicos parámetros expresados por QoS.

Fiedler et al. [FHT10] analiza la relación existente entre estas variables (QoE vs. QoS). Estas relaciones cuantitativas entre QoE y QoS son necesarias para permitir una construcción de mecanismos de control QoE basados en parámetros QoS que se puedan medir. Los autores concluyen que el valor de QoE respecto a la perturbación de QoS se divide en tres regiones separados por dos valores umbrales. En la primera región, el valor de QoE se mantiene estable, incluso con disminución del nivel de QoS. En la segunda región, la disminución de QoS afecta directamente a los niveles de QoE. El usuario percibe una mala calidad pero todavía continúa usando el servicio. En la última región, los niveles tanto de QoS como de QoE son deficientes y el usuario normalmente abandona el servicio.

En cuanto a trabajos de investigación sobre transmisiones de vídeo, Németh et al [NSSG12] propone funcionalidades extra al conmutador OpenFlow. Estas capacidades son: filtros Bloom con reenvío, *Greedy routing* y codificación de la información en la red. Propone asimismo la modificación del protocolo OpenFlow añadiendo tres acciones simples (localizadas en tres registros) soportadas por una lógica XOR entre dos flujos de datos. La codificación se realiza haciendo una copia de cada paquete que se vaya a cifrar enviándola a una cola auxiliar que contiene solamente paquetes cifrados.

Otro trabajo que utiliza SDN para mejorar la calidad en la transmisión de vídeo se presenta en [EDTBT12]. Hilmi *et al* proponen un diseño novedoso de controlador denominado OpenQoS, que también permite implementar calidad

de servicio (QoS) en transmisiones multimedia sobre redes OpenFlow. Para este fin agrupa el tráfico entrante como flujos de datos y flujos multimedia, donde los primeros mantienen la ruta tradicional del camino mínimo mientras que los segundos son dinámicamente situados en rutas donde la calidad de servicio está garantizada. Además de los módulos proporcionados por el controlador Floodlight, OpenQoS añade dos grandes módulos para habilitar la administración de las rutas.

Siguiendo la misma línea de investigación, [GEBMR13] propone una QoE *Fairness Framework* (QFF) con OpenFlow que intenta maximizar la calidad de experiencia QoE del usuario. QFF monitoriza el estado de todos los conmutadores y aplicaciones de vídeo en una red y distribuye dinámicamente los recursos de la red a cada dispositivo.

5.3. Algoritmo de Optimización de la QoE

QoS y QoE son términos que se utilizan a menudo libremente e, incluso a veces, de manera intercambiable [QSE14]. QoS realiza el seguimiento de los componentes discretos de la infraestructura de red tales como servidores, *routers* o el tráfico de red (paquetes IP, flujo de transporte, etc.). Además, la suma de la calidad de los componentes no es igual a la calidad de la suma de componentes. En otras palabras, por tener mejores componentes en una red no significa que se comporte de manera mejor que otra con otros componentes de calidad inferior. La única manera de evaluar la calidad que experimenta un usuario es conectarse al servicio como un usuario. Por lo tanto, los indicadores de rendimiento de QoE están centrados en el usuario a la hora de descargar una página web, acceder a una aplicación, realizar una llamada de teléfono, etc.

El procedimiento para la mejora en la calidad de experiencia del usuario consiste en las siguientes fases:

1. Inicio de funciones básicas del controlador.

2. Identificación de los paquetes de vídeo que circulan por la red.
3. Exploración de la topología de la red.
4. Monitorización de las estadísticas o estado actual de la red.
5. Cálculo del camino óptimo.
6. Reconfiguración de los conmutadores para establecimiento de nueva ruta.

Todas estas fases de las que consta el algoritmo se pueden apreciar en el diagrama de flujo de la Figura 5.1.

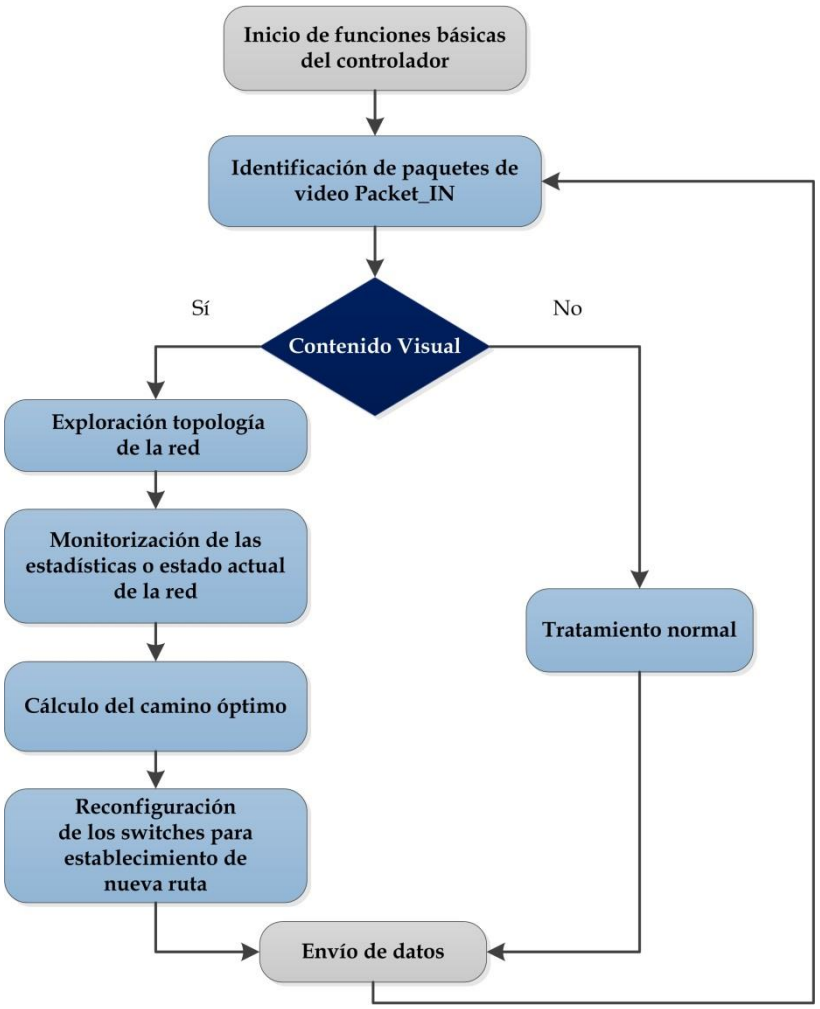


Figura 5.1: Procedimiento de optimización de la QoE.

5.3.1. Inicio de Funciones Básicas del Controlador

Una vez ejecutado Floodlight, se configuran todas las características que se desee tenga el controlador.

Las características que contiene el controlador son:

- Servicios de Floodlight como:
  - ***ITopologyService*:** mantiene la información de la topología que comunica al controlador.
  - ***IDeviceService*:** recorre los dispositivos para saber cómo se encuentran y determina la posición de un dispositivo.
  - ***IRoutingService*:** módulo que calcula rutas entre dos dispositivos dentro de la red OpenFlow.
  - ***ILinkDiscoveryService*:** servicio responsable del descubrimiento y mantenimiento del estado de los enlaces en la red OpenFlow.
  - ***IListeners*:** herramienta de monitorización que cuando recibe un evento se activa y ejecuta la acción que tenga asociada al mismo.
  - ***IThreadPoolService*:** módulo envuelto por un servicio Java de ejecuciones programadas. Puede ser usado para tener hilos de ejecución en tiempos específicos o periódicamente.
  - ***Timers*:** herramienta para lanzar ejecuciones cada cierto período de tiempo.

- **Recepción de PACKET_IN:** se monitorizan todos los paquetes que entran al controlador de este tipo, es decir, paquetes que llegan a un conmutador y no tienen una regla de flujo asociadas a ellos. Este tipo de paquetes puede ser cualquiera de los que componen los mensajes asíncronos descritos en la sección 3.5.1.2.

5.3.2. Identificación de los Paquetes de Vídeo

En este punto el controlador analiza la cabecera del paquete tipo PACKET_IN que entra al controlador y lee el puerto de transporte destino (L4-*Transport Source Port*). Los campos que pueden ser leídos en un paquete se presentan en la Figura 5.2.

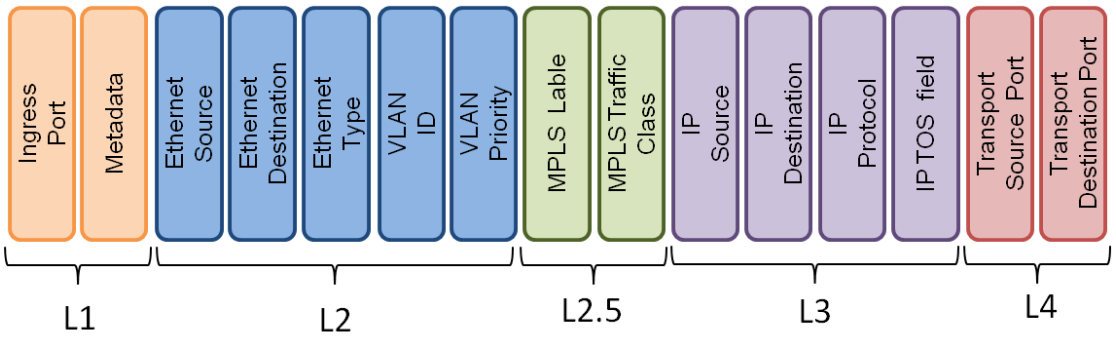


Figura 5.2: Paquete PACKET_IN recibido por el controlador OpenFlow [OSS09] [EDTBT12].

Si el puerto de transporte del paquete es un puerto utilizado para la transmisión de vídeo (en la implementación se utiliza el puerto 5532), el paquete será procesado utilizando el cálculo del camino óptimo para su posterior envío. En caso contrario, el paquete tendrá un procesamiento normal.

5.3.3. Exploración de la Topología

En esta fase se realiza una comprobación de la topología de red para obtener una visión de todos los dispositivos y las conexiones que la componen. Para ello se utilizan los servicios de Floodlight llamados *ILinkDiscoveryService* e *IDevice* (descritos en la sección 5.3.1).

La topología de los enlaces y conmutadores presentados en el plano de infraestructura se representa como un grafo G (N, A) donde N es el conjunto de conmutadores y A es el conjunto de los arcos (enlaces).

Más concretamente, *arc* (*i*, *j*) ∈ A representa el enlace desde el nodo *i* hacia el nodo *j*.

Cada enlace está asociado con un valor llamado *peso del enlace* o *coste del enlace*. En este caso, c_{ij} representa el coste del enlace en el *arc* (*i*, *j*).

El coste del enlace representa un valor fijo como, por ejemplo, saltos ($c_{ij} = 1$), una variación de tiempo de un campo (*jitter*), pérdida de paquetes o el ancho de banda disponible.

R_{st} representa el conjunto de todos los caminos o rutas disponibles desde el conmutador fuente *s* hasta el conmutador destino *t*. En este enfoque se asumen varios supuestos:

- la red es dirigida;
- la red contiene un camino directo desde un conmutador hacia cualquier otro conmutador;
- la red no contiene ciclos negativos.

5.3.4. Monitorización de las Estadísticas y Cálculo del Estado de la Red

Las medidas de rendimiento de una red es un gran desafío.

El *manager* del módulo de estadísticas provisto por NOS puede leer la tasa teórica de datos máxima en un puerto particular a través del parámetro OFP_PORT_FEATURES.

Sin embargo, la capacidad real o tasa de datos máxima depende de las condiciones y calidad de los enlaces entre conmutadores.

Por otra parte, OpenFlow 1.0 especifica contadores en el conmutador en diferentes niveles (tablas, puertos, flujos, colas) pero no provee retardo de

paquetes o medidas de *jitter*.

Además, el controlador lee esta información con el inevitable retardo de tiempo del enlace controlador - conmutador.

Los modelos propuestos en [EDTBT12] [DT13] presentan algoritmos para mejorar el diagnóstico de rendimiento para redes OpenFlow.

Otra solución es una integración de medidas del plano de datos o herramientas especializadas como *sFlow* [FHT10].

En este modelo, el coste del módulo de rendimiento de la red usa el Algoritmo 1:

Primero, el módulo lee la tasa máxima de datos de los enlaces (OFP_PORT_FEATURES) y añade esta información como valor inicial en el coste.

Después, el módulo inicia un temporizador para monitorizar continuamente los contadores de estadísticas de los puertos (OFP_PORT_STATS) y coleccionar periódicamente los bytes enviados.

La división de los bytes enviados respecto al período devuelve la tasa de bytes/segundos. Este valor es actualizado como el vector coste de cada enlace.

Con estos datos se calculará el camino óptimo a seguir (punto 0) por los paquetes que componen el archivo de vídeo.

Algoritmo 1: Función de Rendimiento de la Red

Entradas: grafo red *G(N,A)*
periodo monitorización *t*
ancho de banda del enlace *bw(i,j)* para cada arc*(i,j)*
ajuste factor *a*

Resultado: coste c_{ij} $\forall arc(i,j) \in A$

1. **procedure** FuncionCoste
2. Inicio temporizador *k* con periodo *t*
3. **for each** periodo $k = 0, 1, 2, 3 \ldots . \in t$ **do**
4. **for each** $arc(i,j)$ **do**
5. Lectura de bytes enviados s_i del puerto del enlace inicial
   s_i^k **=** ***tx_bytes*** **in** ***ofp_port_stats(node,port(i))***
6. Lectura de bytes recibidos r_j en el enlace final
   r_j^k**=** ***rx_bytes*** **in** ***ofp_port_stats(node,port(j))***
7. **if** k= 0 (tiempo inicial) **do**
8. $c_{i,j} = 1$
   end if
9. **if** k > 0 (cada periodo**) do**
10. Cálculo de la tasa de datos del enlace
    $dr_{i,j} = \frac{s_i^k - s_i^{k-1}}{t}$
11. Cálculo de los paquetes perdidos del enlace
    $pl_{i,j} = \frac{s_i^k - r_j^{k-1}}{t}$
12. Cálculo del coste del enlace
    $c_{i,j} = 1 + \frac{\propto dr_{i,j} + (1-\propto) pl_{i,j}}{bw_{i,j}}$
    end if
    end for
    end for
    end procedure

5.3.5. Cálculo del Camino Mínimo

Hay múltiples técnicas para proveer QoE en arquitecturas de red típicas *hop-by-hop* [ZhCo10]. Sin embargo, la visión global de la red provista por SDN y la estrategia basada en flujos OpenFlow puede facilitar nuevos modelos de algoritmos eficientes de *routing*.

En este trabajo se presenta una solución básica para QoE *Routing* basado en un grafo de nodos y enlaces. Este procedimiento esta descrito en el Algoritmo 2.

En esta implementación el módulo de QoE *Routing* lee los parámetros QoE provistos por la capa de aplicación a través de la RestAPI. Por ejemplo, el usuario puede establecer una prioridad alta al vídeo *streaming* (obteniendo mejor calidad) respecto a otro tipo de datos a enviar.

Además, este módulo también recibe el valor de coste c_{ij} del enlace devuelto por el módulo de rendimiento de la red al igual que el grafo G (N, A) del módulo de encargado de la topología.

Cuando el controlador recibe un mensaje PACKET_IN, éste analiza la cabecera y establece si el paquete cumple los requisitos QoE.

Si es así, el módulo lee las direcciones IP fuente e IP destino y envía esta información al módulo de cálculo del camino mínimo.

Esta implementación usa el algoritmo de Dijsktra [ZhCo10] para establecer la ruta en la red en función del coste provisto por el módulo de rendimiento de la red. La ruta elegida será el camino con el coste mínimo. Esta ruta será instalada en los dispositivos de red a través del *manager* de flujo de conmutadores. Dicho *manager* usa los mensajes *flowmod* para configurar las tablas de flujo de los conmutadores con mayor prioridad. Floodlight provee el servicio *IRouting* para establecer un camino regular entre conmutadores fuente y destino.

Algoritmo 2: Función QoE Routing

Entradas: grafo red ***G(N,A)***
coste enlace c_{ij} ∀ $arc(i,j) \in A$
switch fuente s, **switch** destino t
parámetros QoS (e.g. TCP/UDP Puerto multimedia)
Salidas: mensajes switch *flowmod* F para apoyar QoS
Resultado: c_{ij} **for each** $arc(i,j) \in A$

1. **procedure** QoS Route
2. packet in recibido del OF-Switch
   $pi \leftarrow PACKET\_IN$
3. **if** pi **fulfil** $QoS\ parameters$ **then**
4. búsqueda de ruta *r* entre fuente y destino usando el coste de enlaces
   $r_{s,t} \leftarrow Dijkstra(G, s, t, c_{i,j})$
5. creación de mensajes *flowmod* F para la ruta r con mayor prioridad
   $F_{s,t} \leftarrow flowmod\_message\_function\ (r_{s,t})$
6. envío de mensajes *flowmod* F a los OF-switches
   $write\_flow\_mod\ (F_{s,t})$
   else
7. procesamiento del paquete *pi* con la mejor estrategia de esfuerzo y menor prioridad
   end if
   end procedure

5.3.6. Reconfiguración de los Conmutadores para Establecimiento de Nueva Ruta

Una vez obtenida la ruta óptima entre el host fuente y el destino se realiza la programación remota de los conmutadores. Este proceso se realiza configurando las tablas de flujo de cada conmutador del camino resultante con un mensaje tipo *flowmod*. En dicho mensaje se establece que todo paquete, cuyo puerto destino alojado en la cabecera sea 5532, se redirija por el puerto calculado previamente en el algoritmo.

Debido a la naturaleza aleatoria de la red, la configuración de los conmutadores no puede ser estática, por lo que los flujos instalados en el conmutador necesitan actualizarse periódicamente. Con este fin, las reglas instaladas en el conmutador tienen un *hard_timeout* constante. Esto quiere decir que, después de t segundos, el flujo es removido del conmutador y se envía un mensaje tipo *flowremoved* al controlador. De esta manera, el controlador recalcula el camino y configura nuevamente los conmutadores.

6. EXPERIMENTOS Y RESULTADOS

En este trabajo de investigación se ha utilizado la versión OpenFlow 1.0.0 [OSS09] [ONF14].

6.1. Herramientas de Desarrollo

Para el desarrollo de este trabajo de investigación se ha utilizado el ordenador y VM especificados en la Tabla 6.1. En ella se emplean los siguientes programas:

- Floodlight 0.90.
- Mininet 2.1.0 - Simulación de redes SDN.
- VLC media player 2.0.8 Servidor y cliente de *video streaming*.

Componente PC	**Especificación**
MacBook Pro	OSX v10.9.4
Procesador	Intel Core i5 2.4 GHz
Memoria RAM	4Gb DDR3
Disco duro	500 Gb
Componente VM	**Especificación**
Sistema operativo	Linux Ubuntu (64 bits)
Memoria base	512 Mb
Memoria video	12 Mb
Nº procesadores	1 procesador

Tabla 6.1: Especificaciones técnicas de las herramientas de simulación.

A continuación se describe las características de las herramientas utilizadas para la simulación.

6.1.1. Mininet

Mininet es un simulador de redes virtuales para SDN. Ejecuta una colección de *host*s, conmutadores con soporte OpenFlow en un *kernel Linux*. Usa virtualización para simular un único sistema y crear topologías de red completas. Un *host* de Mininet se comporta como una máquina real: se puede utilizar el intérprete *ssh* y ejecutar programas. Además, los enlaces de red pueden simular retardos y ancho de banda determinado. Asimismo, se pueden enviar comandos *iperf* para medir el rendimiento de la red. Mininet es un proyecto de código abierto disponible en [Min14].

El simulador Mininet incluye varias características como son:

- Un terminal de línea de comandos (clase *Command Line Interface* (CLI)) para crear redes instantáneas y una API de Python para la creación de redes en las cuales se varían tamaños y topologías. Provee diagnóstico utilizable con comandos como *iperf* y *ping*, a la vez de tener la habilidad de ejecutar un comando a un nodo. Por ejemplo: mininet> h11 ifconfig –a realiza la ejecución del comando ifconfig \-a por parte del nodo h11.
- Los conmutadores OpenFlow pueden ser configurados por medio del protocolo OpenFlow usando controladores externos. De esta manera las aplicaciones de red funcionan automáticamente en una infraestructura real.
- Documentación API completa a través de los *docstrings* de Python utilizando tanto el comando help() como la habilidad de generar documentación PDF/HTML con el comando *make doc*.
- Topologías parametrizadas (subclases Topo) usando el objeto *Mininet*. Por ejemplo, una red con estructura de árbol puede ser fácilmente creada con el comando: mn \-topo tree,depth\=2, fanout\=3.
- Un comando de limpieza para borrar la red virtual (interfaces, procesos,

archivos temporales, etc.) los cuales están alrededor de *Mininet* o *Linux*. Este comando es: mn \-c.

No obstante lo anterior, Mininet tiene algunas limitaciones entre las que se encuentran por ejemplo:

- Los recursos del servidor tienen que dividirse entre todos los *hosts* virtuales, enlaces y conmutadores. Esto implica un error en el rendimiento de la simulación cuando el número de dispositivos simulados es grande.
- Mininet usa un *kernel* único de Linux para todos sus *hosts* virtuales. Esto significa que no puedes ejecutar software que dependa de Windows u otros *kernels* de sistemas operativos.
- Actualmente, Mininet no hace *Network Address Translation* (NAT) fuera de la máquina. Esto significa que los *hosts* virtuales se encuentran aislados y no pueden comunicarse directamente con internet.
- Todos los *hosts* de Mininet comparten el sistema de archivos y el espacio de *Process IDentifier* (PID), por lo que hay que tener cuidado si se está ejecutando *daemons* que requieren configuraciones en el directorio */etc* y no matar procesos erróneos por error.

6.1.2. VLC Media Player

VLC Media Player [VLO14] (comúnmente conocido como VLC) es un reproductor portable, de código abierto, multiplataforma, que además tiene la opción de actuar como servidor *streaming*.

VLC es parte del proyecto *VideoLan*.

El reproductor VLC soporta diferentes métodos de compresión de audio y vídeo y formatos de archivos, incluyendo DVD-Video, video CD y protocolos *streaming* (UDP, HTTP, RTP, RTSP, MMS).

La distribución por defecto de VLC incluye un gran número de decodificadores gratis y librerías para codificar, evitando la necesidad de buscar/calibrar *plugins* propietarios. Algunos de los códecs de VLC están provistos de la librería *libaycodec* del proyecto *FFmpeg*, pero usan principalmente sus propias implementaciones. Además, VLC es el primer reproductor que soporta *playback* de DVDs cifrados en Linux y OS X usando la librería de descifrado de DVDs llamada *libdvdcss*.

VLC posee las siguientes características:

- VLC presenta un diseño modular. De esta manera se facilita incluir módulos, *plugins*, para el soporte de nuevos archivos, códecs o métodos de *streaming*. Actualmente, se disponen de 380 módulos (v.1.0.0). VLC puede reproducir vídeo con formato H.264, MPEG-4, FLV, MMX, D-VHS, HDTV.
- VLC tiene una interfaz estándar basada en módulos, por lo que tiene soporte para consola y múltiples máscaras para interfaz gráfica (Qt 4 para Windows y Linux, Cocoa para OS X y BeAPI en BeOS).
- VLC es un reproductor basado en paquetes; reproduce casi todo los tipos de contenido de vídeo. Puede reproducir éstos, aunque estén dañados, incompletos o no terminados como, por ejemplo, archivos que todavía se están descargando vía *peer-to-peer* (red P2P).

6.2. Simulación

Las pruebas de la topología se realizan a través del simulador de redes Mininet, donde se ejecuta el escenario que envía un archivo de vídeo desde un *host* servidor hasta un *host* cliente a través del protocolo de transmisión de vídeo RTP.

El escenario de la topología, descrito en la Figura 6.1, se compone de cuatro *hosts* virtuales ('h1', 'h2','h3' y 'h4'), cuatro conmutadores ('S1','S2', 'S3' y 'S4') y el controlador desarrollado en este trabajo de investigación.

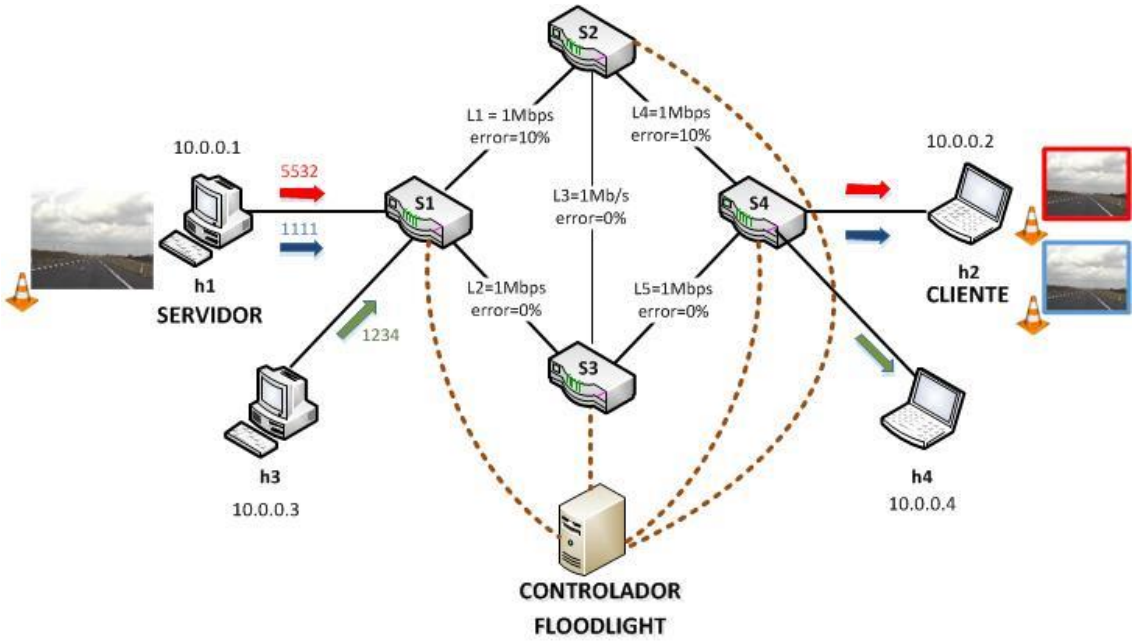


Figura 6.1: Estructura de la topología en las simulaciones.

Los conmutadores están conectados en forma de rombo añadiendo un enlace extra entre S2 y S3.

Por un lado, los *hosts* h1 y h3 están conectados a S1 igualmente que h2 y h4 a S4.

Por otro lado, los enlaces que componen el camino entre los conmutadores S1-S2-S4 contienen un ancho de banda de 1 Mbps y una tasa de pérdida de paquetes del 10%.

Sin embargo, el otro camino posible compuesto por S1-S3-S4 contiene un ancho de banda de 1 Mbps pero sin tasa de error.

El desafío es enviar desde h1 hasta h2 dos vídeos similares al mismo tiempo y comprobar el comportamiento que sufre la red utilizando para uno de ellos el

algoritmo propuesto en este trabajo de investigación y en el otro la función de encaminamiento de Floodlight.

Los datos técnicos del vídeo están descritos en la Tabla 6.2.

Lenguaje	Nombre del Controlador
Códec	MPEG-4 Video (FMP4)
Duración	80 segundos
Tamaño	2.6 Mb
Resolución	352 x 258
Fotogramas totales	2000
Formato decodificado	*Planar* 4:2:0 YUV
Protocolo de transmisión	RTP

Tabla 6.2: Datos técnicos de la simulación.

La simulación comienza cuando el primer vídeo (óptimo) se envía por el puerto 5532 (línea roja), mientras que el segundo se envía por el puerto 1111 definido igualmente como línea azul.

Una vez transcurrido un tiempo *t* se procede a enviar tráfico desde h2 hasta h4 (mostrado con la línea verde) con el fin de saturar la red.

El objetivo es verificar que el vídeo que se recibe en el *host* cliente y que entra por el puerto 5532 es de mejor calidad que el vídeo de 1111 en condiciones extremas.

Esta verificación se realiza mediante el análisis de los vídeos enviados y recibidos midiendo su *Peak Signal-to-Noise Ratio* (PSNR), *Structural Similarity* (SSIM) y *Mean Opinion Score* (MOS) respecto al vídeo original.

Una vez recibidos los paquetes de vídeo en el *host* cliente h2, se guardarán todos ellos en un nuevo archivo para reconstruir el vídeo que emitió el host servidor h1.

Se puede apreciar en la Figura 6.2 la diferencia de calidad entre el vídeo original emitido por el servidor (a) y los dos que recibió el cliente, uno procedente del puerto 5532 (b) con una pequeña pérdida de calidad en algunos píxeles y el otro del 1111 (c) con pérdida de algunos píxeles de la imagen.

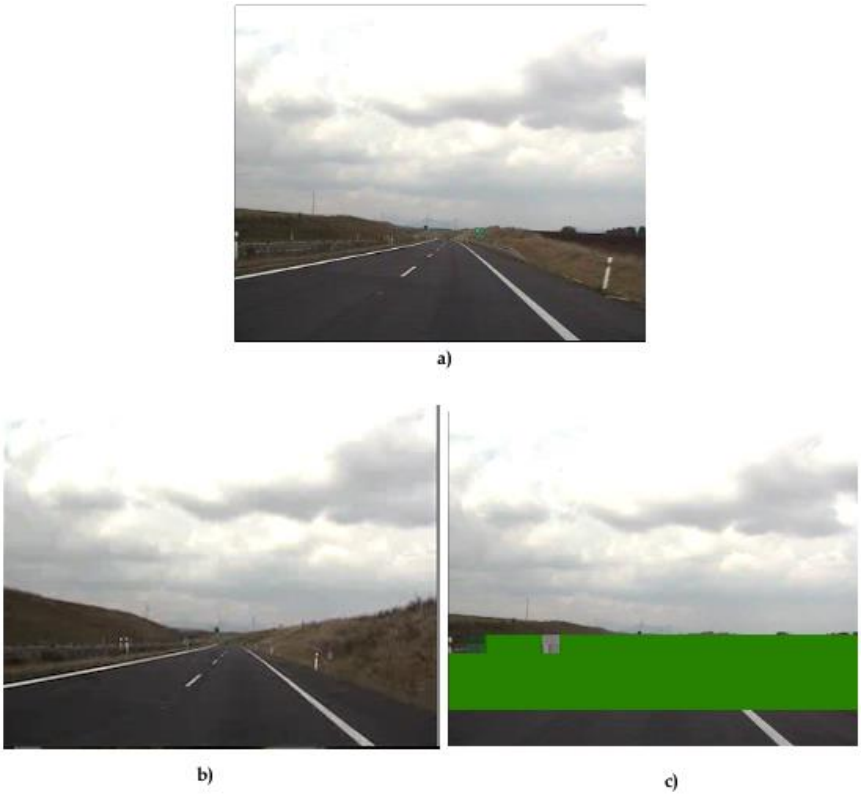


Figura 6.2: a) vídeo original, b) vídeo recibido por puerto 5532, c) vídeo recibido por puerto 1111.

Finalizado el envío, se utilizará una herramienta llamada *Evalvid* [KRW03] para comparar los vídeos recibidos en el cliente respecto al original que se envió desde el servidor con las tres medidas anteriormente mencionadas.

En esta comparación de videos se necesitan los ficheros con extensión *.yuv* que siguen el estándar YUV. Este estándar, conocido anteriormente como YCrCb (Y Cr Cb), es un modelo de representación del color dedicado al vídeo análogo. El parámetro Y representa la luminancia, es decir, información en blanco y negro, mientras que U y V representan la crominancia, es decir,

información con respecto al color.

Una vez obtenidos los ficheros yuv se realiza la obtención de los PSNR del vídeo original respecto al recibido, al igual que la SSIM y MOS.

A partir del archivo donde se encuentra la diferencia entre el vídeo original y el recibido en el *host* cliente, se realizan las gráficas que se describen a continuación.

6.3. Pruebas y Resultados

Para demostrar el rendimiento del controlador descrito en este trabajo de investigación se han realizado diferentes simulaciones para comprobar la calidad del controlador SDN en la transmisión de vídeo.

El proceso consiste en asignar valores distintos al factor α de la función coste del Algoritmo 2. Estos valores influirán en la elección del camino priorizando de forma porcentual la tasa de bytes enviados respecto a los bytes perdidos. Estos valores comienzan en 0 y termina en 1 con incrementos de ¼. De cada una de las simulaciones se obtienen medidas de calidad de vídeo que son PSNR, SSIM y MOS con el programa ya comentado *EvalVid* [KRW03].

6.3.1. Medida PSNR

El método más común para calcular la calidad de un vídeo es el cálculo del *Peak Signal-to-Noise Ratio* (PSNR) fotograma por fotograma [GKKW04]. Derivada de otra medida conocida como es SNR (*Signal-to-Noise Ratio*), el PSNR compara la energía de la señal máxima posible respecto a la energía de ruido en cada fotograma.

En la Ecuación 6.1 se describe el cálculo PSNR para una imagen fuente *s* y la de destino *d* representadas individualmente mediante una matriz con los píxeles que la componen.

$$\mathrm{PSNR}(s,d) = 20 \log \frac{\mathrm{Vpeak}}{\mathrm{MSE}(s,d)} \quad [\mathrm{dB}] \tag{6.1}$$

donde

$$\mathrm{Vpeak} = 2^k - 1$$

siendo k la profundidad del color del bit, y

$$\mathrm{MSE}(s,d) = \frac{1}{n} \sum_{n=1}^{n} (s_i - d_i)^2$$

Las gráficas que se describen en la Figura 6.3 muestran el valor del PSNR a lo largo del eje de abscisas (compuesto de los fotogramas que dura el vídeo) respecto al eje de ordenadas (valor PSNR medido en decibelios (dB)).

En la Figura 6.3 (a) (α igual a 0), la tasa de datos (bytes enviados) no presenta ninguna importancia ya que todo el peso de la función coste recae sobre los bytes perdidos.

En la Figura 6.3 (a) se puede observar que el vídeo con mejor PSNR respecto al vídeo original es el del puerto 5532, seguido del vídeo del puerto 1111 con una diferencia de alrededor de 4 dB. Teniendo en cuenta que la mejora de 3 dB se corresponde con el doble de potencia en la señal, se certifica que el vídeo por el puerto 5532 será de mayor calidad que el del 1111, obteniéndose una menor cantidad de ruido en sus fotogramas.

Por otro lado, se produce un descenso de PSNR cuando la red se satura con el envío de otros datos por el puerto 1234. Pasados unos segundos se observa la recomposición de la señal, dando mayor prioridad al vídeo que circula por el puerto 5532 (que nunca llega al pico inferior del puerto 1111) para aumentar de tal forma su PSNR y, por tanto, mejorando la QoE del cliente.

En la Figura 6.3 (b) donde α es igual a 0.25, 0.75 y 1, se obtienen resultados similares aunque las funciones de coste sean distintas por el porcentaje de importancia de los dos parámetros que la componen como son la tasa de datos

y la tasas de bytes perdidos.

En la Figura 6.3 (b) se puede observar un resultado parecido al del apartado anterior donde el factor era α=0. El vídeo con mejor PSNR respecto al vídeo original es el del puerto 5532, seguido del vídeo del puerto 1111 con una diferencia mayor que los 4 dB de media que teníamos en el apartado anterior.

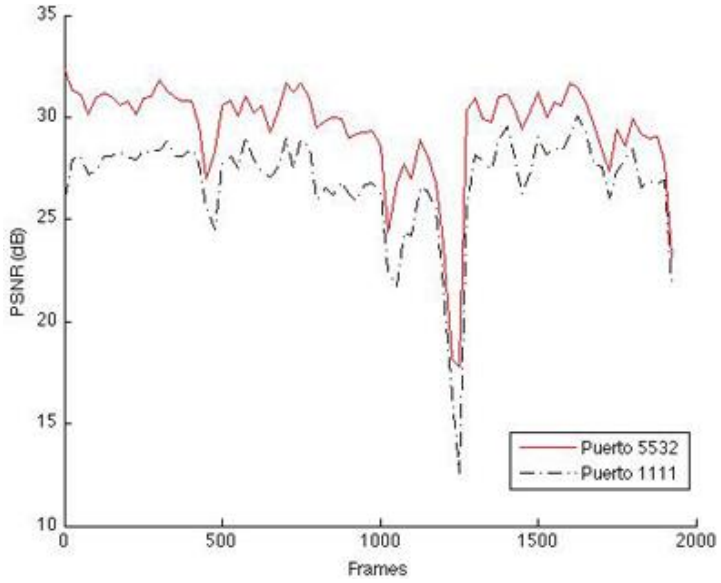
Teniendo en cuenta que la mejora de 1 dB se corresponde con el doble de calidad, se certifica que el vídeo por el puerto 5532 será de mucha mayor calidad que el del 1111 (y que el del apartado anterior), obteniéndose una menor cantidad de ruido en sus fotogramas.

Por otro lado, se produce un descenso de PSNR cuando la red se satura con el envío de otros datos por el puerto 1234. Después del descenso se observa la recomposición de la señal dando mayor prioridad al vídeo que circula por el puerto 5532 (el cual no llega ni a la mitad del descenso del PSNR del puerto 1111) para aumentar de tal forma su PSNR y, por tanto, mejorando la QoE del cliente.

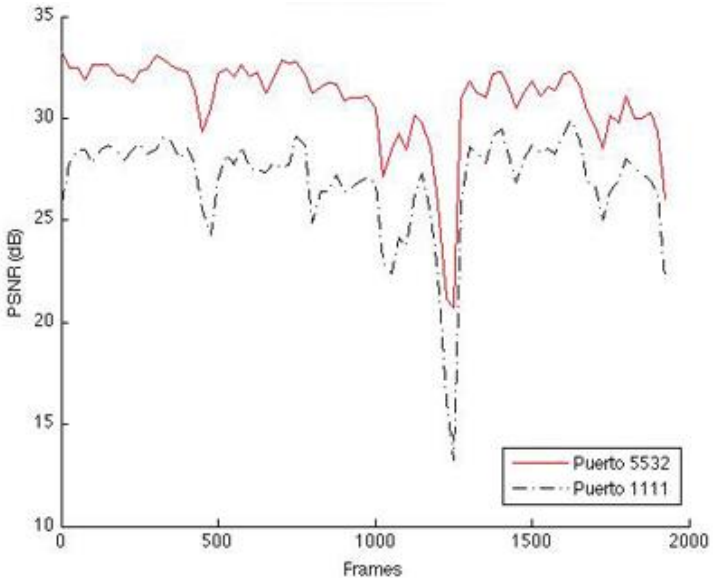
En la Figura 6.3 (c) donde el factor α es 0.5, la función coste está compuesta equitativamente por la tasa de datos y los bytes perdidos.

En la Figura 6.3 (c) se puede observar el resultado de las cinco simulaciones que componen la medida de PSNR. El vídeo con mejor PSNR respecto al vídeo original es el del puerto 5532 seguido del vídeo del puerto 1111 con una pequeña diferencia de alrededor de 1 dB de media.

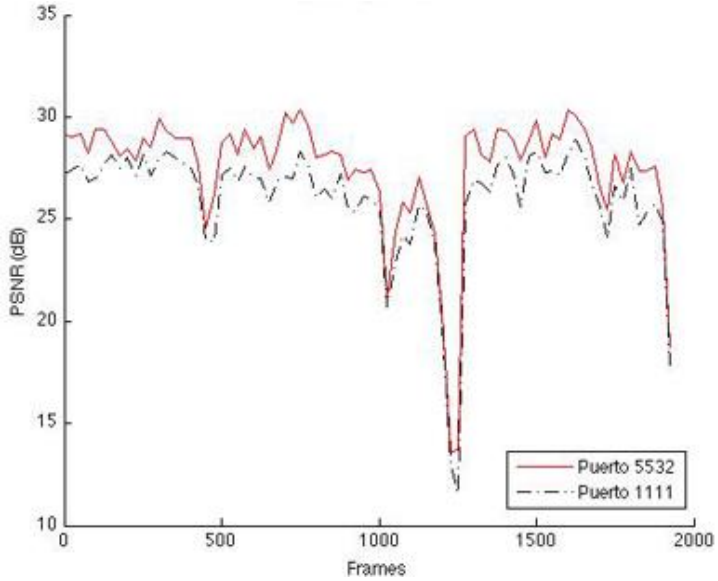
Este caso discrepa con el anterior por su escasa diferencia entre los resultados de los PSNR de los dos puertos. El motivo es que los factores que componen la función de coste devuelven el mismo valor o valores muy parecidos. Por tanto, el camino mejor también puede ser elegido por el vídeo que circula por el puerto 1111, obteniéndose así un resultado muy parecido al del puerto 5532.



(a) α =0



(b) α =0.25



(c) α =0.5

Figura 6.3: PSNR obtenido con valores del factor α de 0, 0.25 y 0.5.

6.3.2. Medida SSIM

El método con el que se mide la semejanza entre dos imágenes es la *Similaridad Estructural* (SSIM) [WRSS04]. En otras palabras, es la medida de la calidad de la nueva imagen tomando como referencia su imagen original no distorsionada y no comprimida. Está diseñada para mejorar los métodos tradicionales de medidas de calidad como PSNR y *Mean Squared Error* (MSE), que son inconsistentes a la percepción del ojo humano. Es tan fácil de calcular como los PSNR pero alberga mayor precisión porque las demás técnicas estiman errores mientras la SSIM no. La información estructural es la idea de que píxeles tienen fuertes interdependencias cuando están especialmente cerca unos de otros.

En la Ecuación 6.2 se muestra cómo se calcula la SSIM respecto a una imagen fuente x y una imagen y representadas individualmente mediante una matriz con los píxeles que la componen.

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_1)} \tag{6.2}$$

donde:

μ_x representa la media de x,

μ_y representa la media de y,

σ_x^2 representa la varianza de x,

σ_y^2 representa la varianza de y

σ_{xy} representa la covarianza de x e y, y

c_1 y c_2 representan dos variables para estabilizar el denominador

Las gráficas de la Figura 6.4 muestran en el eje de abscisas los fotogramas que componen el vídeo mientras que en el eje de ordenadas el valor de la SSIM, comprendido entre 0 y 1.

En los casos donde el factor α es 0 ó 0.5 se obtienen resultados similares pero con la diferencia del peso de la tasa de datos en la función de coste del algoritmo.

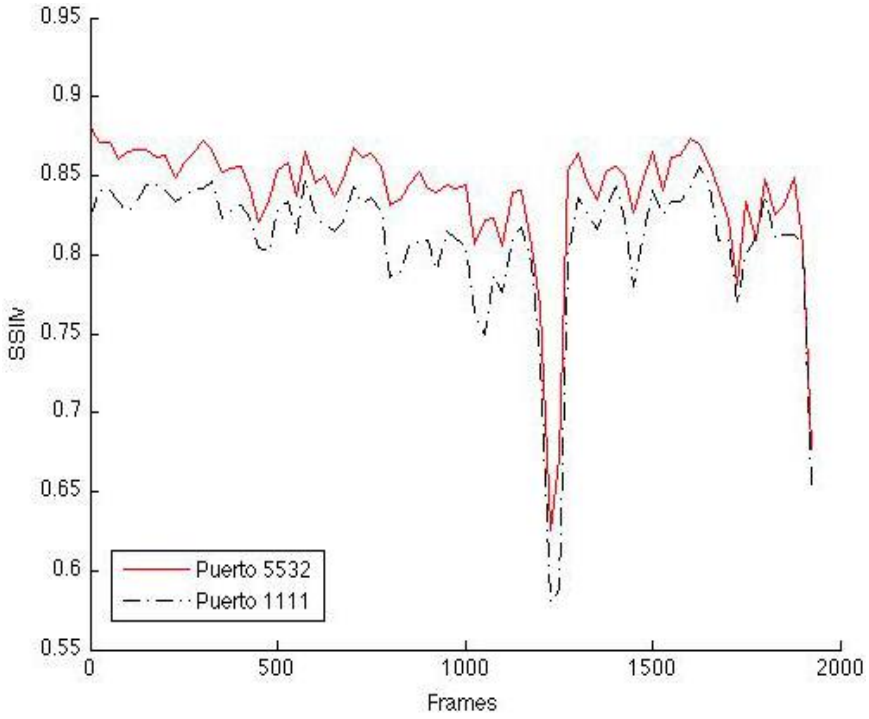
En la Figura 6.4 (a) podemos observar que el vídeo que mejor SSIM tiene respecto al vídeo original es el del puerto 5532 seguido muy de cerca por el del puerto 1111, obteniéndose por momentos el mismo valor entre ellos dos.

Este descenso de SSIM se produce cuando la red se satura con el envío de otros datos por el puerto 1234 pero, como se observa, se recomponen y se da mayor prioridad al vídeo que circula por el puerto 5532 para aumentar de tal forma su SSIM y, por tanto, mejorándose la QoE del cliente.

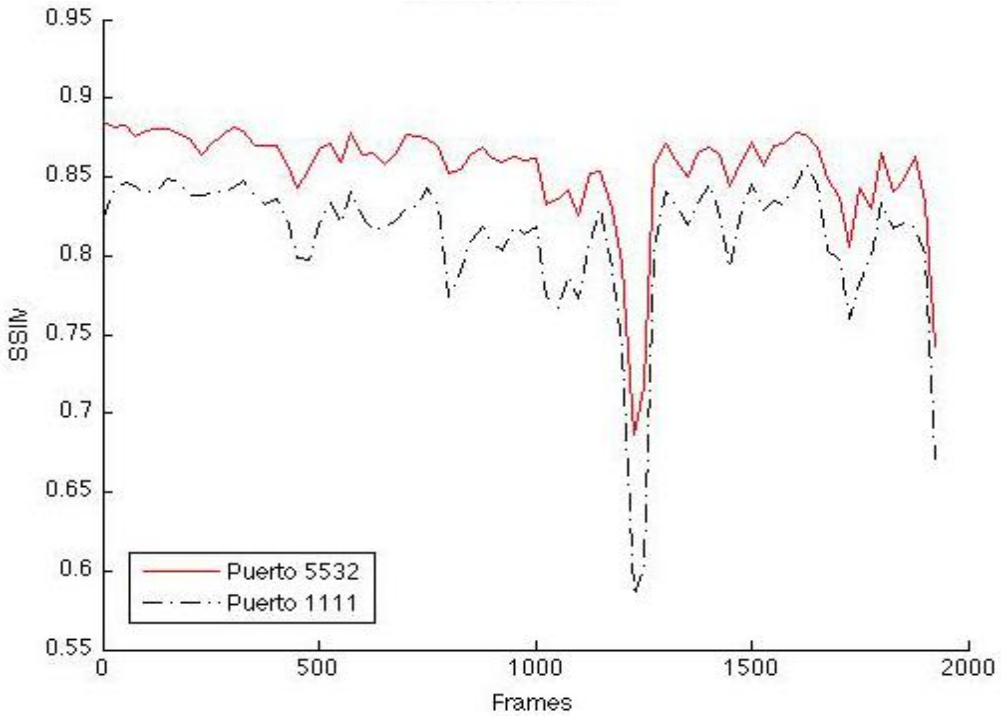
Con estos resultados se verifica que la búsqueda de caminos con la menor tasa de bytes perdidos se obtienen resultados similares a la función *IRouting* de Floodlight del mismo modo que pasaba con la medida de los PSNR.

En la Figura 6.4 (b) se muestran las gráficas correspondientes a tres porcentajes de la función coste con el mismo resultado. Se producen cuando los valores del factor α son 0.25, 0.75 y 1, obteniéndose resultados similares, como se muestra en la Figura 6.4 (a), pero con la diferencia del peso la tasa de datos.

El promedio de valores del puerto 5532 es menor que el del puerto 1111 a excepción del pico a los 1250 *frames* respecto al inicio. Esto indica que las imágenes que componen el vídeo del 5532 comparten una mayor similaridad estructural respecto a las del vídeo original ya que sus valores no oscilan entre picos tan separados.



(a) α =0



(b) α =0.25

Figura 6.4: Resultado SSIM de simulación con valores del factor **α** de 0 y 0.25.

6.3.3. Medida MOS

La medida que describe la calidad de un vídeo frente al ojo humano es la MOS (*Mean Opinion Score*) mediante una métrica de calidad subjetiva [GKKW04]. Dicha métrica está compuesta por los valores que se muestran en la Tabla 6.3.

Grado	Calidad	Descripción
1	Malo	Muy distorsionado
2	Pobre	Distorsionado
3	Medio	Ligeramente distorsionado
4	Bueno	Perceptible
5	Excelente	Imperceptible

Tabla 6.3: Correspondencia grado – calidad.

Los umbrales de tiempo de reacción para la percepción del ser humano son conocidos como [FHT10]: 100 ms es el límite aproximado en el cual un usuario humano experimenta que el sistema reacciona instantáneamente; menor que 1 s el usuario sigue manteniendo una buena experiencia pero el retardo es apreciable; menor que 10 s el usuario empieza a perder la atención del vídeo, mientras que si se rebasan dichos 10 s hay un riesgo de que el usuario abandone la actividad.

Ya que no hay una correspondencia directa entre PSNR y MOS [GKKW04], existen relaciones heurísticas de sus valores como se muestran en la Tabla 6.4.

PSNR (dB)	MOS	PSNR (dB)
< 20	1 (Malo)	< 20
20 - 25	2 (Pobre)	20 – 25
25 - 31	3 (Medio)	25 – 31
31 - 37	4 (Bueno)	31 – 37
> 37	5 (Excelente)	> 37

Tabla 6.4.: Correspondencia grado - calidad

Los resultados de las gráficas que se describen a continuación muestran el porcentaje de los valores PSNR de cada uno de los puertos en el eje de abscisas (compuesto por los 5 rangos de calidad MOS) respecto al eje de ordenadas (porcentaje utilizado de PSNR dentro de cada rango).

En la Figura 6.5 se puede observar como los vídeos que recibe el cliente contienen una calidad MOS en todos sus rangos inferior al mejor calidad (MOS = 5).

La mayor concentración de valores PSNR (alrededor del 85% de ellos) se encuentran en el rango de calidad media por lo que los vídeos tendrán una ligera sensación de distorsión respecto al vídeo original.

En este caso se muestran los resultandos procedentes de los factores 0.25, 0.75 y 1, que son similares. En el primer caso, se le da más porcentaje a la tasa de datos que a los bytes perdidos, mientras que en el segundo y tercero se le da mayor importancia a la información perdida durante el envío.

En la Figura 6.6 se puede observar como los vídeos que recibe el cliente contienen una calidad MOS en todos sus rangos menos en el de calidad media (MOS = 3). La mayor concentración de valores PSNR del puerto 5532 se encuentran en el rango de mejor calidad, mientras que en el puerto 1111 se concentra en el rango de calidad media.

En este caso donde el factor α es de 0.5, la tasa de datos y de bytes perdidos contribuyen de manera equitativa al cálculo de la función coste.

En la Figura 6.7 se puede observar como los vídeos que recibe el cliente contienen una calidad MOS en todos sus rangos de forma similar.

La mayor concentración de valores MOS se encuentran en los rangos de baja calidad por lo que los vídeos tendrán una sensación de distorsión respecto al vídeo original.

Por último, se compara con cuál de todos los factores se obtiene una mejor calidad de vídeo respecto al ojo humano.

En la Figura 6.8 se muestran los valores MOS de cada uno de los puertos respecto a los factores que comprenden la función de coste del módulo que calcula el camino óptimo del vídeo.

En la misma podemos observar que el factor que experimenta menos diferencias entre los dos puertos sucede para $\alpha = 0.75$. Esto ocurre porque proporciona mayor porcentaje a la tasa de datos que a los bytes perdidos, por lo que la calidad de los vídeos recibidos en ellos será bastante parecida.

Por otro lado, el valor más alto de todos los factores es cuando $\alpha = 1$ (sólo tiene en cuenta los bytes perdidos). En este factor el puerto 5532 tiene mayor diferencia respecto al puerto 1111 en los demás factores. Con esto se certifica que los bytes perdidos son fundamentales para obtener un gran valor PSNR.

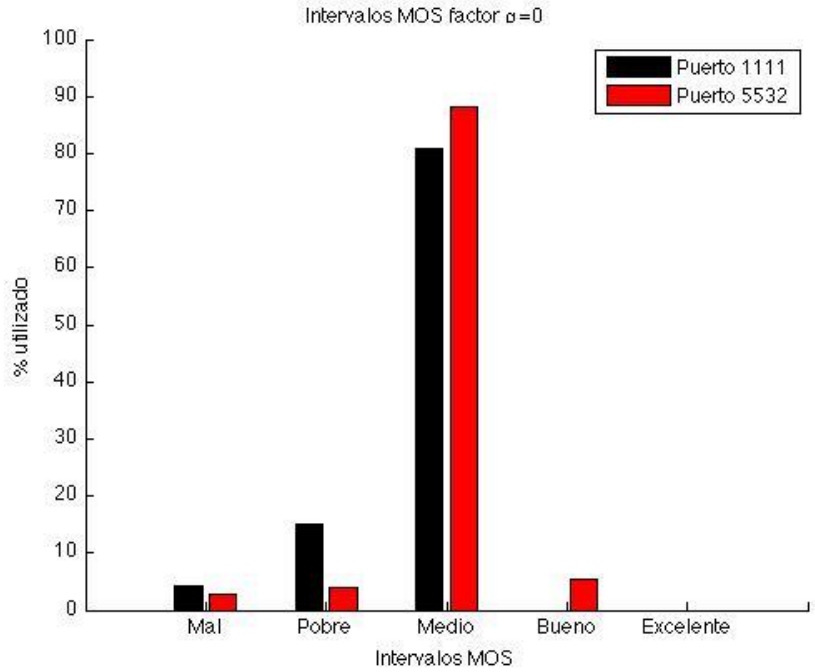


Figura 6.5: Resultado MOS de simulación con factor α =0.

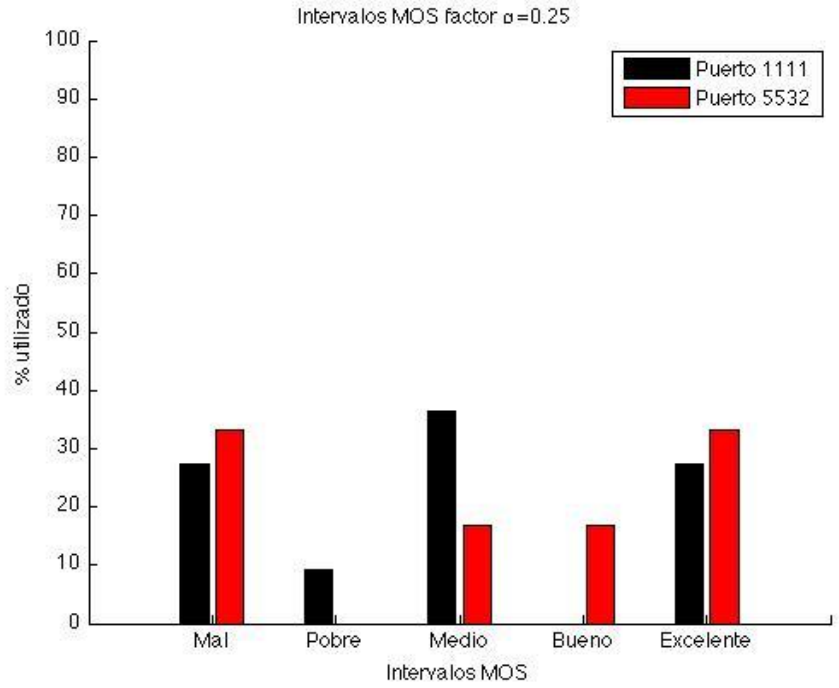


Figura 6.6: Resultado MOS de simulación con factor α =0.25.

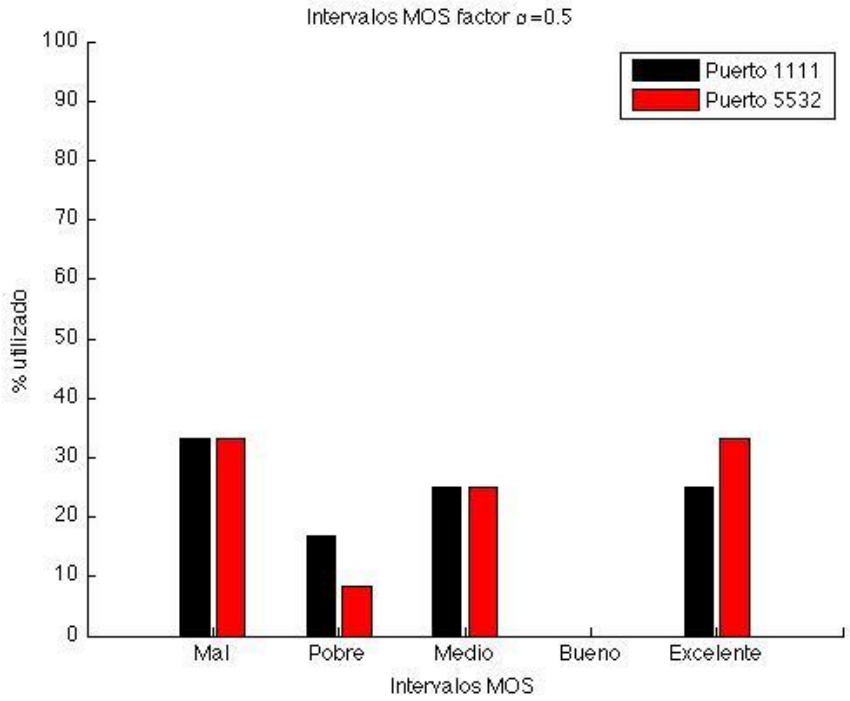


Figura 6.7: Resultado MOS de simulación con factor α =0.5.

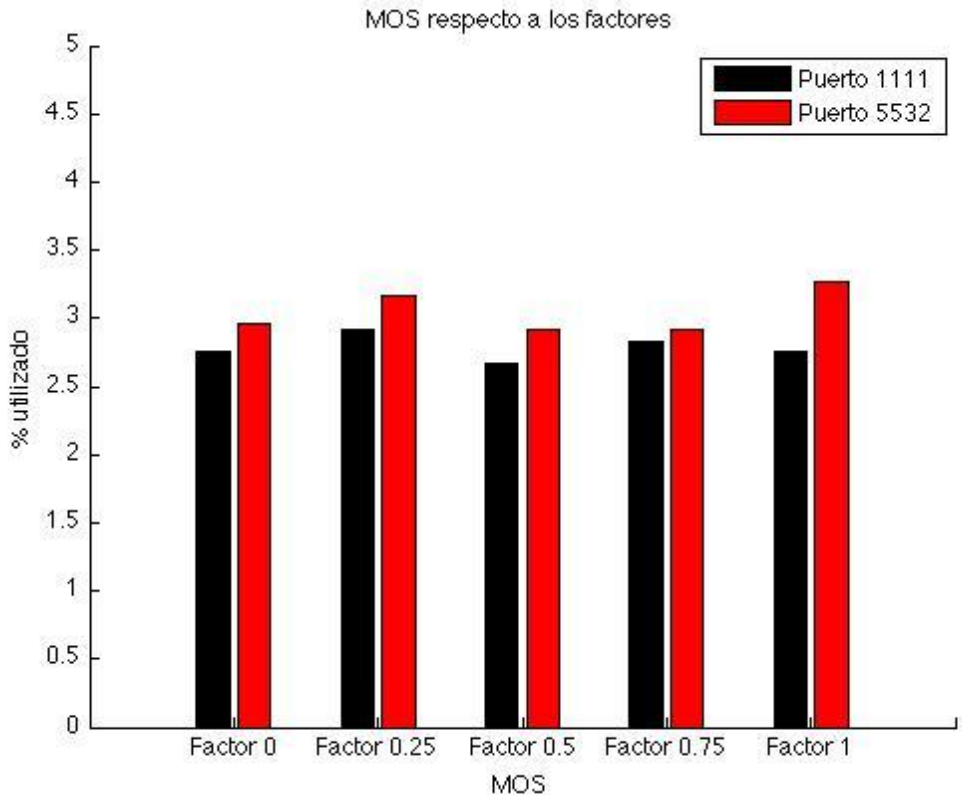


Figura 6.8: Comparación medida MOS respecto a todos los factores.

7. CONCLUSIONES Y TRABAJO FUTURO

7.1. Conclusiones

Las tecnologías de red se han convertido en un elemento crítico en prácticamente todas las actividades humanas. El número de dispositivos así como la cantidad de tráfico transportándose a través del Internet continúa creciendo exponencialmente. Sin embargo, el desarrollo de nuevos protocolos y servicios (movilidad, QoS, transmisión de vídeo digital) ha sido limitado, principalmente por la fuerte unión entre el hardware especializado en la transmisión de paquetes y sistemas operativos ejecutando cientos de protocolos estáticos (algunos de ellos privados) y permitiendo al administrador de red únicamente la configuración del equipo de red. Asimismo, el tiempo de desarrollo de una nueva idea desde su diseño, simulación, pruebas, publicación en un estándar y finalmente instalación en los equipos de red puede tomar algunos años. Claramente, las redes de nueva generación necesitan un cambio con respecto a la rigidez presente en las actuales arquitecturas.

La arquitectura actual de Internet tiene grandes problemas de soporte, verificación y cumplimiento de seguridad. Los típicos sistemas de seguridad están basados en la protección de los host (antivirus) o en la instalación de dispositivos de red especializados como *middleboxes* que detectan anomalías en la red. Sin embargo estas soluciones se vuelven ineficaces cuando el host está comprometido o por continuas actualizaciones por parte del administrador de red. En este punto, SDN puede ayudar a mejorar y desarrollar nuevos modelos de seguridad.

Uno de los problemas relacionados con el control centralizado es la vulnerabilidad de la red. Un fallo en el controlador puede afectar negativamente a la resistencia de la red y por tanto comprometiendo la

información que está viajando en ella. Actualmente se está trabajando en arquitecturas de backup para que el controlador siga dando servicio en caso de fallos. Actualmente, la configuración de cada uno de los controladores de respaldo es posible pero no el mecanismo conexión y comunicación entre ellos. De igual manera, debido a que el control se encuentra centralizado, la posibilidad de ataques de denegación de servicio es mayor.

Debido a las grandes expectativas de los usuarios, se ha necesitado de despliegues rápidos de nuevas aplicaciones, calidad de servicio y seguridad en el acceso a ellas. Sumándole a lo anterior, tenemos las nuevas tendencias de virtualización y Cloud computing que fuerzan a los data centers tradicionales a producir un cambio.

Muchas arquitecturas que utilizan estos data centers son muy complejas, costosas y rígidas para los entornos de computación tan dinámicos que existen actualmente. Por tanto, los administradores de red ganan en flexibilidad y movilidad incluyendo SDN a dichos centros de datos mejorando así el Cloud computing.

Por su parte la virtualización persigue el aislamiento de múltiples redes lógicas, cada una de ellas con sus propios mecanismos de direccionamientos y reenvío pero compartiendo la infraestructura física. SDN provee la oportunidad de alcanzar la capa de abstracción de un hardware.

Software Defined Networking (SDN) es una arquitectura innovadora que propone la separación entre el plano de datos y plano de control, permitiendo su independiente evolución y desarrollo. Actualmente, la principal cristalización de SDN es OpenFlow, una arquitectura abierta diseñada inicialmente para realizar experimentos en redes heterogéneas, sin afectar el tráfico de usuarios reales. Openflow reúne las capacidades homogéneas del hardware de red actuales y las libera. La especificación OpenFlow establece las reglas de comunicación entre el plano de datos y un plano de control

centralizado y permite el control del comportamiento de toda la red a través de aplicaciones de software (API). *Open Networking Foundation* (ONF) reúne aproximadamente a 90 empresas y su tarea es la publicación, promoción y adopción de la especificación OpenFlow [OSS09]. El uso de OpenFlow [OSS09] quita la limitación de la rigidez presente en los protocolos estáticos, abre la posibilidad de innovar rápidamente y permite a la comunidad científica el desarrollo de nuevos paradigmas en tecnologías de red.

Este trabajo de investigación busca la mejora de calidad QoE en la transmisión de video streaming desde un host cliente a otro destino en una red SDN utilizando el protocolo OpenFlow. Durante la realización de este trabajo de investigación, se ha implementado un controlador SDN basado en Floodlight en el que se ha dado prioridad a los videos que circulan por la red identificados por el puerto de envío, concretamente el 5532. La característica básica de este controlador es mejorar la QoE en la búsqueda del camino óptimo entre un dos switches para maximizar todo lo posible la calidad de video en la recepción de éstos por parte del cliente. Este camino es calculado mediante una variante del algoritmo de Dijkstra el cual busca el camino más corto entre dos puntos. La elección de un camino u otro es posible por una función que vincula un valor a cada enlace de la red. Este valor está compuesto por unos parámetros extraídos una colección de estadísticas calculadas previamente cada cierto intervalo de tiempo. Las simulaciones que se han realizado han querido llevar al extremo el comportamiento a la red mediante la saturación de sus enlaces con los videos a enviar. Los resultados de las pruebas verifican la mejora de calidad de este algoritmo respecto al servicio de Floodlight mediante tres medias de calidad de video como: PSNR, SSIM y MOS.

7.2. Trabajo Futuro

En la actualidad, SDN se ha posicionado en gran lugar para ocupar las redes del futuro. A pesar de ofrecer múltiples ventajas con respecto a arquitecturas

tradicionales, existen retos pendientes para completar su implementación en redes comerciales.

En primer lugar se encuentra el encaminamiento entre diferentes dominios SDN y la coexistencia con arquitecturas tradicionales. Para solventar este problema es interesante proponer una estructura jerárquica de controladores de tal manera que en vez de la comunicación entre los dos que rigen cada sistema autónomo, tener un nivel superior con otro que maneje estos dos. De esta forma el retardo aumentaría ligeramente por el incremento de mensajes con este nuevo controlador pero se compensaría con la escalabilidad que proporcionaría este modelo.

Además, las compañías despliegan una amplia variedad de mecanismos de seguridad en sus data centers para proteger sus aplicaciones de posibles ataques. Estos mecanismos son empleados a menudo con otras aplicaciones que realizan balanceo de carga, almacenamiento en caché y aceleración de aplicaciones. El aislamiento de tráfico y control de acceso para los usuarios finales se encuentran entre las múltiples políticas de reenvío que se deben cumplir.

Por otro lado, el talón de Aquiles de esta tecnología es la caída o fallo del controlador SDN. Siguiendo la línea de fallos en la red, ¿Qué pasaría si falla un enlace? Este desafío es muy importante en las comunicaciones multimedia y QoE (por ejemplo audio o video streaming). El fin que se persigue es recuperar el camino de red cuando uno o varios enlaces han caído. Actualmente se ha conseguido solventar este problema con un controlador que incorpora un mecanismo de recuperación de red con un camino alternativo cuando se produce un fallo. Por otra parte se ha llegado a predecir o anticipar un posible fallo antes de que se produzca en la red basándose en la obtención de dos caminos disjuntos (en el caso que falla uno, utilizar el otro sin que haya ningún problema en la comunicación). El retardo que se produce calculando los dos

caminos disjuntos iniciada ya la transmisión, se podría solventar calculando caminos alternativos junto con el camino principal antes de proceder al envío.

El simulador Mininet es una potente herramienta para comprobar los protocolos o mecanismos en vías de construcción para SDN. Presenta algunos problemas que se podrían mejorar. Uno de ellos es la estabilidad en la simulación cuando se están manejando grandes cantidades de enlaces como ocurre en la topología en forma de malla. De igual modo, se debería revisar funciones predefinidas por falta de escalabilidad en el tamaño de las topologías. Un ejemplo es la función que calcula el ancho de banda (iperf) puesto que se demora en su ejecución incluso acabando en fallo.

Respecto al controlador Floodligt, se deberían incorporar algunas funciones predefinidas su API como por ejemplo una que obtenga el ancho de banda actual de un enlace en concreto. Otra característica importante que se debería añadir es la obtención de mayor cantidad de información de las propiedades de la topología que se asignaron en el script que se ejecuta procedente de Mininet.

Por último, el algoritmo propuesto en este trabajo de investigación no está cerrado y acaba aquí. Se podrían incluir varias características adicionales a él aumentando de tal forma su eficacia. Una extensión sería incluir o tener en cuenta los valores de delay, jitter, throughput en la función de coste para la obtención del camino mínimo. Otra podría ser usar nuevos módulos para predecir descenso de rendimiento en algún enlace, módulos de replanificación de ruta en caso de que algún enlace caiga, proporcionar seguridad frente a ataques de denegación del servicio.

BIBLIOGRAFÍA

[AAKS98] D. Alexander, W. Arbaugh, A. Keromytis, and J. Smith, "A Secure Active Network Environment architecture: Realization in SwitchWare," IEEE Network, vol.12, no.3, pp.37,45, May/Jun 1998.

[Ca99] K. Calvert. "Architectural Framework for Active Networks Version 1.0". 1999

[CCFRS05] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and Implementation of a Routing Control Platform," in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2, NSDI'05, (Berkeley, CA, USA), pp. 15–28, USENIX Association, May 2005.

[CFPL07] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, (New York, NY, USA), pp. 1–12, ACM, August 2007.

[CNP03] A. Courtney, H. Nilsson, and J. Peterson, "The Yampa Arcade," in Proceedings of the 2003 ACM SIGPLAN workshop on Haskell, (New York, NY, USA), pp. 7–18, ACM, August 2003.

[COU13] Control and Data Together, Software Defined Networking course, https://www.coursera.org/

[DT13] K. Dhamecha and B. Trivedi "SDN Issues – A Survey ", International Journal of Computer Applications, vol. 73, No.18, July 2013.

[E13] D. Erickson, “The Beacon Openflow Controller,” in Proceedings of the Second ACM SIGCOMM Workshop on Hot topics in Software Defined Networking, (New York, NY, USA), pp. 13–18, ACM, August 2013.

[EDTBT12] Egilmez, Hilmi E and Dane, S Tahsin and Bagci, K Tolga and Tekalp, A Murat “OpenQoS: An OpenFlow Controller Design for Multimedia Delivery with end-to-end Quality of Service over Software-Defined Networks" in Asia-Pacific Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), pp. 1-8, December 2012.

[ELIP14] European Lighthouse Integrated Project, Internet of Things Architecture, http://www.iot-a.eu/public

[FBMP12] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, “A Replication Component for Resilient OpenFlow-based Networking,” in Proceedings of the 2012 IEEE Network Operations and Management Symposium, pp. 933–939, IEEE, April 2012.

[FGR13] N. Foster, A. Guha, M. Reitblatt, A. Story, M. Freedman, N. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison, “Languages for Software Defined Networks,” IEEE Communications Magazine, vol. 51, pp. 128–134, February 2013.

[FHFM11] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, “Frenetic: A Network Programming Language,” in Proceedings of the The 16th ACM SIGPLAN International Conference on Functional Programming, (New York, NY, USA), pp. 279–291, ACM, September 2011.

[FHT10] Fiedler, M.; Hossfeld, T.; Tran-Gia, P., "A Generic Quantitative Relationship between Quality of Experience and Quality of Service," Network, IEEE, vol.24, no.2, pp.36, 41, March-April 2010.

[FO14] Frenetic Organization, https://www.frenetic-lang.org/pyretic.

[FRZ13] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," 2013.

[GBMP13] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," Future Generation Computer Systems - The International Journal of Grid Computing and Science, vol. 29, pp. 1645–1660, September 2013.

[GEBMR13] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards Network-wide QoE Fairness using Openflow-assisted Adaptive Video Streaming," in Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking, FhMN '13, (New York, NY, USA), pp. 15, August 2013.

[GHM05] A. Greenberg, G. Hjaimtysson, D. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A Clean Slate 4D Approach to Network Control and Management," ACM SIGCOMM Computer Communication Review, vol. 35, pp. 41–54, October 2005.

[GKKW04] J Gross, J Klaue, H Karl, A Wolisz, Cross-layer optimization of OFDM transmission systems for MPEG-4 video streaming, Computer Communications, Volume 27, Issue 11, 1 July 2004, Pages 1044-1055, ISSN 0140-3664.

[GKPC08] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," ACM SIGCOMM Computer Communication Review, vol. 38, pp. 105–110, July 2008.

[GRF13] A. Guha, M. Reitblatt, and N. Foster, "Machine-verified Network Controllers," ACM SIGPLAN NOTICES, vol. 48, pp. 483–494, June 2013.

[GYAC09] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A Network Virtualization Layer," tech. rep., OpenFlow Switch Consortium, October 2009.

[GYAC10] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the Production Network be the Testbed?," in Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, (Berkeley, CA, USA), pp. 1–6, USENIX Association, October 2010.

[HSM12] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement problem," in Proceedings of the First Workshop on Hot Topics in Software Defined Networks, (New York, NY, USA), pp. 7–12, ACM, August 2012.

[KF13] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," IEEE Communications Magazine, vol. 51, pp. 114–119, February 2013.

[KRW03] J. Klaue, B. Rathke and A. Wolisz "EvalVid – A Framework for Video Transmission and Quality Evaluation", 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, pp. 255-272, Urbana, Illinois, USA, September 2003.

[KSDM12] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely, "Towards QoE-driven Multimedia Service Negotiation and Path Optimization with Software Defined Networking," in Proceedings of the 20th International Conference on Software, Telecommunications and Computer Networks, vol. 1, pp. 1–5, IEEE, September 2012.

[KSXFE11] H. Kim, S. Sundaresan, M. Chetty, N. Feamster, and W. K. Edwards, "Communicating with Caps: Managing Usage Caps in Home Networks," ACM SIGCOMM Computer Communication Review, vol. 41, pp. 470– 471, August 2011.

[LNRS04] T. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The Softrouter Architecture," in Proceedings of ACM SIGCOMM Workshop on Hot Topics in Networking, November 2004.

[MABP08] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," ACM SIGCOMM Computer Communication Review, vol. 38, pp. 69–74, March 2008.

[MFHW12] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A Complier and Run-time System for Network Programming Languages," ACM SIGPLAN NOTICES, vol. 47, pp. 217–230, January 2012.

[MGBC09] L. A. Meyerovich, A. Guha, J. Baskin, G. H. Cooper, M. Greenberg, A. Bromfield, and S. Krishnamurthi, "Flapjax: A Programming Language for Ajax Applications," ACM SIGPLAN NOTICES, vol. 44, pp. 1–20, October 2009.

[Min14] Cbench, https://github.com/mininet.

[MRFRW13] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, “Composing Software Defined Networks,” in Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, (Berkeley, CA, USA), pp. 1–14, USENIX Association, April 2013.

[NRFC09] A. Nayak, A. Reimers, N. Feamster, and R. Clark, “Resonance: Dynamic Access Control for Enterprise Networks,” in Proceedings of the 1st ACM workshop on Research on enterprise networking, (New York, NY, USA), pp. 11–18, ACM, August 2009.

[NSSG12] F. Németh, A. Stipkovits, B. Sonkoly, and A. Gulyás, “Towards SmartFlow: Case Studies on Enhanced Programmable Forwarding in OpenFlow Switches," ACM SIGCOMM Computer Communication Review, vol. 42, pp. 85, August 2012.

[OMC13] OpenFlow Management and Configuration Protocol (OF-Config) v.1.1.1,” pp. 1–172, March 2013.

[ONF14] Open Networking Foundation”, https://www.opennetworking.org/

[OSS11] OpenFlow Switch Specification v.1.1.0, pp. 1–56, 2011.

[PaPe12] S. M. Patrick Le Callet and A. Perkis, “Qualinet White Paper on Definitions of Quality of Experience,” European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003), March 2012.

[PF14] Project Floodlight, http://www.projectfloodlight.org/floodlight/

[PFCMC10] P. S. Pisa, N. C. Fernandes, H. E. Carvalho, M. D. Moreira, M. E. M. Campista, L. H. M. Costa, and O. C. M. Duarte, “OpenFlow and Xen-Based Virtual Network Migration,” in Communications: Wireless in Developing Countries and Networks of the Future, vol. 327, pp. 170–181, Springer Berlin Heidelberg, September 2010.

[Pl14] PlanetLab, https://www.planet-lab.org/

[PPAC09] B. Pfaff, J. Pettit, K. Amidon, and M. Casado, “Extending Networking into the Virtualization Layer,” in Proceedings of ACM SIGCOMM HotNets, ACM, October 2009.

[PWH13] K. Pentikousis, Y. Wang, and W. Hu, “MobileFlow: Toward Software- Defined Mobile Networks,” IEEE Communications Magazine, vol. 51, pp. 44–53, July 2013.

[QSE14] QoS vs. QoE, http://www.witbe.net/technologie/qos-vs-qoe/

[RFRS12] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, “Abstractions for Network Update,” ACM SIGCOMM Computer Communication Review, vol. 42, pp. 323–334, October 2012.

[RMTF09] A. Ramachandran, Y. Mundada, M. B. Tariq, and N. Feamster, “Securing Enterprise Networks Using Traffic Tainting,” tech. rep., August 2009.

[SJSZRP00] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge, “Smart packets: Applying Active Networks to Network Management," ACM Transactions on Computer Systems, vol. 18, pp. 67, February 2000.

[SJSZRP98] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partridge, “Smart Packets for Active Networks," 1999 IEEE Second Conference on Open Architectures and Network Programming, March 1998.

[SSCF13] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, “Are We Ready for SDN? Implementation Challenges for Software-Defined Networks,” IEEE Communications Magazine, vol. 51, pp. 36–43, July 2013.

[SSCP12] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, “A Demonstration of Fast Failure Recovery in Software Defined Networking,” in Testbeds and Research Infrastructure. Development of Networks and Communities, vol. 44, pp. 411–414, Springer Berlin Heidelberg, June 2012.

[SSI14] SENSEI, Integrated EU Project, http://www.ict-sensei.org/

[SZMM13] L. Shi, B. Zhang, H. T. Mouftah, and J. Ma, “DDRP: An Efficient Data-driven Routing Protocol for Wireless Sensor Networks with Mobile Sinks,” International Journal of Communication Systems, vol. 26, pp. 1341–1355, October 2013.

[TZE13] The Zettabyte EraTrends and Analysis, tech. rep., CISCO, May 2013.

[VBG13] A. Valdivieso, L. Barona and L. Villalba, “Evolution and Challenges of Software Defined Networking”, in Proceedings of 2013 Workshop on Software Defined Networks for Future Networks and Services, pp. 61-67, IEEE, November 2013.

[VGSKF13] P. Vlacheas, R. Giaffreda, V. Stavroulaki, D. Kelaidonis, V. Foteinos, G. Poulios, P. Demestichas, A. Somov, A. R. Biswas, and K. Moessner, "Enabling Smart Cities through a Cognitive Management Framework for the Internet of Things," IEEE Communications Magazine, vol. 51, pp. 102 – 111, June 2013.

[VKF12] A. Voellmy, H. Kim, and N. Feamster, "Procera: A Language for High- Level Reactive Network Control," in Proceedings of the First Workshop on Hot topics in Software Defined Networks, (New York, NY, USA), pp. 43–48, ACM, August 2012.

[VLO14] VideoLan Organization, http://www.videolan.org/index.es.html.

[VW12] A. Voellmy and J. Wang, "Scalable software defined network controllers," ACM SIGCOMM Computer Communication Review, vol. 42, no. 4, pp. 289–290, August 2012.

[WoTu01] T. Wolf and J. Turner, "Design Issues for High-performance Active Routers," IEEE Journal on Selected Areas in Communications, vol. 19, pp. 404–409, March 2001.

[WRSS04] Z. Wang, A. Bovik, H. Sheikh, Eero Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity", IEEE Transactions on Image Processing, vol. 13, no. 4, April 2004

[YDAG04] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework." RFC 3746 (Informational), April 2004.

[ZhCo10] T. S. E. N. Zheng Cai, Alan L. Cox, "Maestro: A system for scalable openflow control," tech. rep., December 2010.